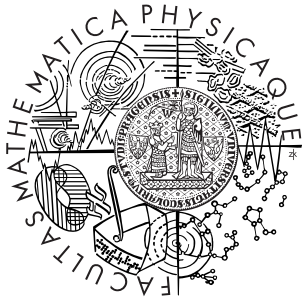
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lukáš Kerpl

Systém pro hlasové ovládání programu Amarok

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. Jiří Semecký, PhD.
Studijní program: Informatika, Programování

2009

Na tomto místě bych rád poděkoval svému vedoucímu Jiřímu Semeckému za jeho rady a připomínky a především za nápad skvělého tématu, díky čemuž mě práce bavila většinu nad ní stráveného času.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 25. května 2009 Lukáš Kerpl

Obsah

Název práce: Systém pro hlasové ovládání programu Amarok
Autor: Lukáš Kerpl
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: RNDr. Jiří Semecký, PhD.
e-mail vedoucího: jirka@google.com

Abstrakt: V předložené práci studujeme ovládání hudebního přehrávače Amarok pomocí lidského hlasu. Problém komunikace s počítačem pomocí hlasu je rozdělen na několik částí. V první části musíme rozpoznat, co uživatel říká. V poslední části bychom měli být schopni uživateli odpovědět, případně mu poskytnout nějaké informace. Mezi těmito dvěma částmi, musíme ovládat program, v našem případě hudební přehrávač Amarok. Tato práce popisuje všechny části a jejich spojení do fungujícího celku. Nechybí ani popis programu stejně jako popis použitých knihoven, včetně možných alternativ. Kapitola Využití a další možný vývoj obsahuje zamyšlení nad přínosem tohoto programu a ovládání počítače pomocí hlasu a to jak v současné době, tak především v budoucnosti.

Klíčová slova: hlas řeč Amarok rozpoznávání ovládání syntéza KDE

Title: Amarok Speech Control
Author: Lukáš Kerpl
Department: Institute of Formal and Applied Linguistics
Supervisor: RNDr. Jiří Semecký, PhD.
Supervisor's e-mail address: jirka@google.com

Abstract: In the presented work we study voice control of Amarok music player. The problem of controlling computer using speech is divided into several parts. In the first part, we need to recognize what user says. In the last one, we should be able to answer to the user or give him some information. In between of these parts, we need to control the program, in our case the Amarok media player. In this work we describe all of these parts, as well as theirs conjunction into working program. Description of the usage of this program and even description of used libraries including potential alternatives is included. The last chapter, *Usage and future development*, considers the contributions of this program and computer voice control in general at the current time and in the future.

Keywords: voice speech amarok recognition synthesis control TTS ASR KDE

Kapitola 1

Úvod

Vývoj ve světě počítačů jde závratnou rychlostí kupředu, procesory jsou čím dál rychlejší a mají více jader, operační paměti je více a více, stejně tak kapacita disků roste. I aplikace jsou stále větší, umí více věcí a rozhodně nenechávají výpočetní výkon ležet ladem. Dalo by se říci, že vše je rychlejší, větší, pohodlnější... no prostě lepší. Jedna věc se však již dlouhá desetiletí nemění, jde o způsob komunikace s počítačem pomocí klávesnice, myši a monitoru. Proto mě již na první pohled zaujalo téma ročníkového projektu.

Ačkoliv jsem o práci s hlasem na počítači nevěděl takřka nic, to téma mě nadchlo. Velmi mě zaujala představa programu, se kterým si člověk může povídat. Projekt se navíc jevil jako dobrá možnost vyzkoušet, jaké jsou možnosti posunout vývoj dopředu i v oblasti ovládání počítače, poodhalit tajemství toho, proč stále používáme klávesnici a myš, proč si s počítačem nepovídáme.

Stejný cíl jako jsem měl já, při volbě ročníkového projektu má i tato práce. Chtěl bych zde popsat své zkušenosti s psaním takového programu, využitím existujících projektů a jejich spojení ve funkční celek.

Text je rozdělen do několika kapitol, ve kterých rozebírám různá témata související s projektem. Hned v následující kapitole stručně představím vlastní projekt, jeho vlastnosti, požadavky na systém i trochu historie jeho vývoje. Tato kapitola by měla umožnit udělat si představu o vytvořeném projektu a umožnit čtenáři lepší pochopení následujících kapitol.

V kapitole třetí se věnuji všem použitým technologiím. Každou z nich krátce představím. Pohovořím o možných alternativách, jejich výhodách a nevýhodách a samozřejmě o svém rozhodnutí a důvodech, které jsem pro něj měl. Každý, kdo by chtěl napsat hlasem ovládaný program, nebo takto

nějaký program upravit, bude stát na podobných křižovatkách jako já. Je zde tedy možnost najít inspiraci pro to, kterou cestou se vydat.

Stejně inspirativní by měla být i část nazvaná Implementace, kde se věnuji tomu, jak jsem jednotlivé externí programy a knihovny spojil dohromady ve fungující celek.

Když je program hotov, zbývá ho otestovat a když vše funguje, můžeme ho začít používat. Využití programu se věnuje samostatná kapitola. V ní se pokouším vysvětlit používání programu, popisuji, které příkazy je možné používat a jak na ně bude program reagovat. Zmíním také doporučení, jak program využít co nejlépe.

Po dokončení programu, chvíli po tom, co nás přejde nadšení z toho, že program opravdu funguje, přichází chvíle, kdy se zamyslíme, co dál. Takové zamyšlení obsahuje předposlední kapitola a věnuji se v ní jak současnému, tak možnému budoucímu využití program a jeho rozšíření. Zabývám se zde i možností podobných programů a jejich smyslu či užitečnosti.

V závěrečné kapitole se pokouším shrnout a ohodnotit výsledek svého snažení. Zhodnotím co jsem psaním práce získal, pokusím se odhadnout co získá uživatel mého programu a co může získat potencionální čtenář této práce.

Můj program jsem pojmenoval Ascon (Amarok Speech Control) a pod tímto jménem ho můžete nalézt na serveru sourceforge.net, kde jsou kromě instalačního archivu dostupné i zdrojové texty a dokumentace. Stejný obsah má i přiložené CD, které obsahuje i elektronickou podobu této práce.

Kapitola 2

Popis programu

Následující kapitola obsahuje základní fakta o programu Ascon. V krátkosti je zde shrnuta historie mé práce, jaká jsem činil rozhodnutí a jakým problémům jsem musel čelit.

První část se věnuje výběru programovacího jazyka, do které se promítla volba knihovny pro rozpoznávání řeči. Dalším tématem je výběr knihovny pro rozpoznávání řeči a systému pro syntézu řeči.

Druhá část popisuje základní údaje o používání programu, jaké jsou licenční podmínky, jaké programy a knihovny jsou potřeba pro běh programu a další podmínky pro správné fungování programu. Více informací o používání programu, popis instalace a ovládání programu je popsáno v kapitole Používání programu.

2.1 Základní technické informace

Při psaní doplňků k již existujícímu programu je programátor tímto programem velmi ovlivněn. Nejen, že musí dodržovat určité rozhraní, určené pro komunikaci doplňku s vlastním programem, ale i tím, v jakém prostředí a na jaké platformě je program spouštěn a v neposlední řadě také zvyklostmi jeho uživatelů.

A i když jsem si to z počátku plně neuvědomoval, byl jsem nakonec donucen se těmto požadavkům a zvyklostem plně podřídit. Původně měl program být napsán v jazyce C++, jednak kvůli rychlosti, jednak proto, že ve stejném jazyce byla napsána i knihovna pro rozpoznávání řeči, v podstatě nejdůležitější část systému.

Jenže vývoj ve světě počítačů letí neúprosně dopředu a tak další verze

knihovny pro rozpoznávání řeči už byla v jazyce Java. Rané verze mého doplňku byli napsané v C++, ale ukázalo se, že to není šťastné řešení. Rozmanitost Linuxových distribucí a rozšířenost přehrávače Amarok, respektive Linuxu obecně (zde je třeba poznamenat, že existují i porty přehrávače Amarok pro Windows a Mac OS X), po různých hardwarových platformách do značné míry znemožnili snadnou distribuci doplňku. Amarok jako součást prostředí KDE patří do světa obyčejných uživatelů a tam se překlad ze zdrojových kódů nehodí. A tak jsem se musel přizpůsobit světu, pro který jsem chtěl tvořit a věřím, že to byla dobrá volba.

Po konzultaci s vedoucím jsem tedy dospěl k názoru, že celý program přepíši do jiného jazyka. Abych se vyhnul nutnosti kompilace pro rozličné systémy, rozhodl jsem se, že program napíši v nějakém skriptovacím jazyce. Vývojáři přehrávače Amarok doporučují na svých stránkách Ruby, a protože ani další kandidáty jako Python, či Perl jsem neznal, neváhal jsem a vybral si jako programovací jazyk Ruby. Musel jsem se s ním nejprve seznámit a tak se projekt trochu pozdržel, ale nakonec mi tento velmi stručný a přímý jazyk i hodně času ušetřil. O tom se podrobněji zmíním v kapitole Použité technologie.

Díky zvolenému jazyku je program velmi malý co se týče místa na disku i délky zdrojového kódu. Je rozdělen do několika souborů. Navíc běží na jakémkoli systému, kde je nainstalován interpret jazyka Ruby, který se v základní instalaci vyskytuje ve většině linuxových distribucích, a v ostatních systémech je možné jej snadno doinstalovat, tím ideálně pokrývá všechny systémy, kde se používá přehrávač Amarok.

Dále bylo třeba zvolit správně jak rozpoznávat řeč a jak uživateli zpět odpovídat. Pro rozpoznávání řeči byla volba jasná. Projekt CMU Sphinx je v dnešní době jediný živý "open source" projekt pro rozpoznávání řeči. Naprosto opačná situace byla při volbě systému pro syntézu řeči. Projektů zabývajících se syntézou řeči existuje několik. První mě samozřejmě napadl ten nejznámější a nejpoužívanější. Festival. Nakonec jsem ale kvůli pohodlí uživatelů zvolil systém KTTS, který sice sám syntézu neprovádí, ale umí použít ostatní projekty, jako je Festival.

2.2 Použití programu

Program je ke stažení serveru *www.sourceforge.net*. Jde o svobodný software pod licencí GNU GPL a je tedy možné jej libovolně používat, šířit a upravovat.

2.2 Použití programu

Ke stažení je program dodán v připraveném komprimovaném balíčku takové struktury, jakou vyžaduje Amarok pro instalaci doplňků a je tedy možné jej snadno pomocí správce doplňků v Amarok přímo nainstalovat a poté i spustit.

Jediným požadavkem pro běh programu je dostupnost použitých knihoven, tedy KTTS pro převod textu na řeč, CMU Sphinx pro rozpoznání řeči a samozřejmě interpret jazyka Ruby.

Pro správné fungování je také třeba vlastnit funkční mikrofon a reproduktory či sluchátka. Po spuštění doplňku program projde vaši kolekci hudby v přehrávači Amarok a sestaví gramatiku pro rozpoznávání, to může chvíli trvat pokud je kolekce velmi rozsáhlá. Poté už jen čeká na hlasové příkazy a vykonává je. Navíc sleduje a hlásí změny přehrávaných stop v přehrávači.

Pro plné využití všech funkcí je také vhodné, mít v hudební kolekci informace o obsažených skladbách. Tento požadavek je vysvětlen v poslední části kapitoly Používání programu, která se jmenuje Doporučení pro použití.

Kapitola 3

Použité technologie

V této kapitole se budu věnovat použitým technologiím. Zmíním zde programovací jazyk Ruby, který jsem se naučil, jeho hlavní přednosti pro využití psaní doplňků hudebního přehrávače Amarok. Zmíním také možné alternativy Python a Perl, které také doporučují tvůrci hudebního přehrávače Amarok jako vhodné pro psaní doplňků.

Dále představím použitou knihovnu pro rozpoznávání lidské řeči CMU Sphinx. Popíši, co celý projekt CMU Sphinx obsahuje a jaké části jsou použity v mém řešení.

Další nezbytnou součástí mého projektu je syntéza řeči. O té si povíme v kapitole Festival a KTTS. Zmíním i konkurenci a možné alternativy v této oblasti, kterých není málo.

Nakonec se budu věnovat protokolu DCOP a jeho nástupci DBUS.

3.1 Ruby

Ruby je dynamický, interpretovaný, skriptovací programovací jazyk, jehož autorem je Yukihiro "matz" Matsumoto. Hlavní filozofií programovacího jazyka Ruby je však snaha o přirozenost. Cílem autora bylo vytvořit jazyk jednoduchý na čtení a přirozený pro vyjádření programátorových myšlenek. Přesto jde o jazyk velmi komplexní, nechybí možnost zpracování výjimek, či práce s vlákny.

Ruby je také jazykem objektovým. Vše v Ruby je objekt, to můžeme ukázat na příkladu, který se často uvádí jako příklad přirozenosti zápisu v Ruby:

Ukázka 3.1: Třikrát hello v Ruby

```
3.times { puts "hello" }
```

Takto vypíšeme třikrát slovo "hello" na standardní výstup.[1] Velkou výhodou a velkým plusem při rozhodování pro použití byla i možnost volání systémových příkazů jejich zapsáním do zpětných apostrofů.

Alternativ v podobě ostatních programovacích jazyků bylo hodně. Důležitým faktorem bylo, aby šlo o jazyk, který nepotřebuje kompilaci a vzhledem k doporučení od tvůrců hudebního přehrávače Amarok jsem výběr zúžil pouze na tři doporučené programovací jazyky: Ruby, Perl a Python.

Perl jsem zavrhl z čistě subjektivního důvodu. Pokud jsem se měl naučit nový programovací jazyk, chtěl jsem se naučit něco moderního. Zbyl tedy Python a Ruby.

Python je rozšířenější, ale žádnou zásadní výhodu proti Ruby jsem nenašel a tak pro Ruby rozhodla pro mě čitelnější a jednodušší syntaxe a jednodušší ovládání přehrávače Amarok. Pro představu je zde srovnání přiřazení jména aktuálně přehrávané skladby do proměnné artist v Ruby a v Pythonu (příklad je převzat z *http://amarok.kde.org/wiki/Script-Writing_HowTo*).

Ukázka 3.2: Ruby: jméno umělce

```
artist = `dcop amarok player artist`
```

Ukázka 3.3: Python: jméno umělce

```
from dcopext import DCOPClient, DCOPApp
import sys

# create a new DCOP-Client:
client = DCOPClient()

# connect the client to the local DCOP-server:
client.attach()

# create a DCOP-Application-Object to talk to amarok:
amarok = DCOPApp('amarok', client)

# call a DCOP-function:
ok, artist = amarok.player.artist()
```

3.2 CMU Sphinx

CMU Sphinx je sada nástrojů pro rozpoznávání řeči, kterou vyvíjejí na Carnegie Mellon univerzitě v Pittsburghu. Projekt obsahuje společnou knihovnu Sphinx-base, čtyři dekodéry mluvené řeči (Sphinx-2, Sphinx-3, Sphinx-4 a PocketSphinx), nástroj pro trénování akustických modelů SphinxTrain, dva nástroje pro trénování jazykových modelů (cmuclmtk a SimpleLM) a dvě další utility (cepview a lm3g2dmp).

Ve svém projektu používám dekodér mluvené řeči Sphinx-4, který v současné době představuje, alespoň podle autorů projektu, nejvyspělejší systém pro rozpoznávání řeči. Sphinx-4 je napsán celý v programovacím jazyce Java. Původně měl být projekt Sphinx-4 pouhým přepsáním Sphinx-3 do programovacího jazyka Java, nakonec byl ale přepracován a velmi vylepšen. Na vývoji Sphinx-4 se, kromě týmu z Carnegie Mellon univerzity, podíleli i společnosti Hewlett Packard, Sun Microsystems, Mitsubishi, nebo Kalifornská univerzita v Santa Cruz a dokonce i světoznámý Massachusettský technologický institut (MIT)[2].

Sphinx-4 používá k rozpoznávání řeči metodu skrytých markovovských modelů. Je to jedna ze dvou metod používaných pro rozpoznávání řeči. Druhou metodou je dynamické borcení času, ale od ní se postupně upouští, protože metoda skrytých markovovských modelů přináší mnohem lepší výsledky.

V současné době projekt CMU Sphinx v podstatě nemá konkurenci mezi volně šiřitelnými programy pro rozpoznávání lidské řeči. Většina projektů, které nalezneme na internetu, je mrtvá, vývoj ustal. Z těchto starších projektů nemůže žádný CMU Sphinx konkurovat už jen proto, že vývoj v této oblasti stále pokračuje.

3.3 Festival a KTTS

Festival je systém pro syntézu řeči vyvinutý centrem pro výzkum hlasových technologií (CSTR) na univerzitě v Edinburghu. Systém je možné využít mnoha způsoby: programem Festival pro příkazovou řádku, nebo jako interpret příkazů napsaný v jazyce Scheme, C++ knihovnu, z Javy, nebo dokonce z prostředí editoru Emacs[3].

Festival je možné použít pro syntézu mnoha jazyků. V základní verzi podporuje americkou a britskou angličtinu, welštinu a španělštinu. Systém je ale možné doplnit o syntézu dalších jazyků. Existuje sada programů a

návod, jak vytvořit vlastní syntetický hlas pro Festival. Díky tomu lze na internetu nalézt mnoho dalších jazyků pro systém Festival, včetně projektu Festival Czech, který přidává podporu češtiny.

KTTS (KDE Text To Speech System) je systém pro syntézu řeči pro prostředí KDE. Umožňuje využít několik systémů pro syntézu řeči a hlavně obsahuje příjemné uživatelské GUI pro nastavení systému a jeho vlastností jako je výška hlasu nebo rychlost řeči.

KTTS implementuje protokol DCOP, což umožňuje snadné použití z ostatních programů.

Na rozdíl od systému pro rozpoznávání řeči, srovnatelných systémů pro syntézu řeči existuje několik. Návrh programu umožňuje snadnou změnu systému pro syntézu řeči za jiný. Za zmínku stojí především systém Epos (Epos Speech Synthesis System), vyvíjený Ústavem fotoniky a elektroniky Akademie věd České republiky a systém MBROLA, který je sice zdarma, ale jeho zdrojový kód není otevřený. Proto byl nakonec použit nejvíce rozšířený Festival.

KTTS využívající Festival nakonec dostalo přednost před samotný Festivalem ze dvou důvodů. Prvním je, že KTTS je součástí prostředí KDE a tedy umožňuje nastavení v grafických dialozích zapadajících do prostředí KDE. Druhým důvodem je, že pokud někdo používá jiný systém pro syntézu řeči než Festival, stačí mu nastavit KTTS na svůj systém.

3.4 DCOP

DCOP je zkratkou pro Desktop Communication Protocol, což je název jednoduchého protokolu pro komunikaci mezi programy. Zjednodušeně lze říci, že se jedná o dálkové ovládání programů. V tomto smyslu je použit i v mé práci, kde můj program využívá tento protokol k ovládání přehrávače Amarok a systému KTTS.

Protokol DCOP podporuje většina aplikací, které jsou součástí grafického prostředí KDE ve verzi 3. Aplikace umožňují pomocí toho protokolu provádět vlastní akce, aniž by bylo třeba využít jejich vlastní ovládací rozhraní. Tak je například možné, bez kliknutí na tlačítko stop, zastavit přehrávač hudby nebo změnit pozadí plochy bez otevření příslušného dialogu.

Jeho jednoduchost napovídá, že bude velmi nenáročný na systémové prostředky, jak se můžeme přesvědčit nahlédnutím do dokumentace[4], podle provedených testů vytvoření DCOP klienta navýší požadavky aplikace o 100kB

paměti, což je velmi nízká cena za použití tohoto protokolu, zvláště pokud si uvědomíme, kolik paměti potřebují dnešní běžné aplikace.

DCOP má velmi jednoduchý model, kde každá aplikace používající DCOP ke komunikaci je klient. Všechny aplikace spolu komunikují přes DCOP server, který od všech aplikací zprávy přijímá a doručuje je adresátům.

Existují dva typy zpráv. Jeden je pro vyžádání si informace, kdy aplikace vyšle požadavek a čeká na příchod odpovědi. Druhý pouze informaci odešle, to je užitečné pro odeslání příkazu, nebo odpovědi na nějaký požadavek. V mém programu jsou použity oba typy.

DCOP protokol je postaven nad protokolem ICE (Inter Client Exchange), který je součástí standardu X11R6. Kromě grafické knihovny Qt, která tvoří základ KDE, nemá žádné další závislosti na knihovnách, což umožňuje jeho široké použití v KDE aplikacích.[4]

V nové, čtvrté verzi grafického prostředí KDE, je DCOP nahrazen protokolem DBUS, který jako standard připravila společnost freedesktop.org a je protokolem DCOP hodně inspirován.

3.5 DBUS jako nástupce DCOP

DBUS je nástupcem protokolu DCOP v prostředí KDE4. Jde o standard od společnosti freedesktop.org, který popisuje komunikaci mezi programy.

Bohužel v současné době Amarok 2 implementuje funkce podle MPRIS 1.0 (Media Player Remote Interfacing Specification), což neumožňuje pomocí protokolu DBUS například ovládat hudební kolekci přehrávače Amarok. Díky tomu není možné snadno upravit můj projekt tak, aby fungoval i v přehrávači Amarok 2.

Tento nedostatek je při psaní doplňků pro Amarok možné obejít tak, že se přistupuje k objektům přehrávače pomocí Qt API. To ale vyžaduje použití programovacího jazyka JavaScript a tak je nutné existující doplňky zcela přepsat. Tento přístup navíc přímo závisí na uživatelském rozhraní a není tak odolný proti jeho změnám.

Díky tomu je tedy možné, že od druhé verze přehrávače Amarok již nebude možné doplněk používat.

Kapitola 4

Implementace

Samotný program se skládá z několika částí. Řešení obsahuje samostatný program pro rozpoznávání příkazů v lidské řeči, ten je napsán, stejně jako knihovna pro rozpoznávání řeči kterou používá, v jazyce Java. Další částí je skript v jazyce Ruby, který pro tento program tvoří gramatiku popisující přijímané příkazy v lidské řeči. Hlavní program, který tyto programy využívá, se také skládá z několika částí. Rozdělením na části a jejich fungováním se budeme věnovat v následující kapitole.

4.1 Program recognize

Program recognize se nachází v souboru "recognize.jar". Program je napsán v jazyce Java a slouží jako obal ke knihovně CMU Sphinx. Program má jedinou, zato však velmi důležitou funkci. Tou je převádět mluvené příkazy na text, tak aby se dali dále zpracovávat. Program poslouchá zvuk přicházející z mikrofónu a rozpoznaný text zapisuje do pojmenované roury. Z té si potom text vyzvedává objekt třídy listener v hlavním programu.

Příkazy k rozpoznávání jsou definovány gramatikou, uloženou v souboru "recognizer.gram". Jeho generování má na starost skript generate_gram, kterému se v této kapitole také budu věnovat.

Samotný program je velmi jednoduchý. Po importování potřebných knihoven, hlavně těch z projektu CMU Sphinx, a přečtení konfigurace ze souboru, se vytvoří objekty microphone a recognizer důležité pro zpracovávání zvuku z mikrofonu a rozpoznávání příkazů.

Po alokaci prostředků potřebných pro rozpoznávání se program pokusí začít poslouchat vstup mikrofonu. Pokud se to nepovede, skončí. Pokud ano,

vstoupí do nekonečné smyčky, ve které se na objektu recognizer volá metoda recognize. Tady se program zastaví do doby, než zaznamená zvuk na mikrofonu. Pokud se tento zvuk podařilo rozpoznat pomocí naší gramatiky, je výsledek převeden na text pomocí metody getBestResultNoFilter a zapsán do pojmenované roury, aby jej mohl hlavní program dále zpracovat. V případě, že se zvuk nepodařilo rozpoznat, vypíše program chybovou hlášku.

4.2 Třída Listener

Třída Listener obsluhuje přijímané příkazy. Je součástí hlavního programu napsaného v jazyce Ruby a nalezneme ji v souboru "listener.rb". Třída má na starost vytvoření pojmenované roury, ze které bude přijímat textové příkazy a spuštění programu recognize, který tuto rouru bude plnit příkazy.

Třída má konstruktor, který se stará o vytvoření pojmenované roury a inicializaci proměnných. Důležitou proměnnou je objekt třídy Amarok, pojmenovaný player, který slouží pro ovládání přehrávače. Boolská proměnná not_listen značí, jestli se má poslouchat příkaz pro doplněk, nebo jde o odpověď na požadavek o upřesnění uživatelova výběru. Tato odpověď je uchována v proměnné answer_buffer. Další proměnné, listening_thread a recognizer_thread, odkazují na vlákna, ve kterých je spuštěna funkce listen_in_thread a program recognizer. Poslední dvě proměnné, mutex a cv_answer slouží k synchronizaci vláken a ještě je zmíním na konci této části.

Destruktor třídy se stará o odstranění pojmenované roury a zničení vláken listening_thread a recognizer_thread.

Nejdůležitější metodou třídy Listener je funkce listen_in_thread. Je zavolána z metody listen, která nedělá nic jiného, než že v novém vlákně spustí právě funkci listen_in_thread. Samotná funkce obsahuje nekonečnou smyčku, ve které se čte vstup z pojmenované roury. Po přečtení příkazu z pojmenované roury se zkontroluje jestli jde skutečně o příkaz, nebo o odpověď na naši otázku uživateli. Pokud jde o příkaz (proměnná not_listen je false), je pomocí větvení case určeno o jaký příkaz se jedná a ten je pak proveden pomocí volání patřičné metody objektu player.

Pokud je příkaz "play me ..." může dojít k situaci, že není jednoznačné, co se má přehrát. To ale zjistíme, až ve třídě Collection, respektive její metodě add_to_playlist. Pokud k tomu dojde, je volána metoda get_answer třídy Listener, která vrací text, který je odpovědí na otázku kladenou programem uživateli. Metoda get_answer nastaví proměnnou not_listen na

true (nebude poslouchat příkazy, ale odpověď na otázku) a zavolá metodu get_buffer. Metoda get_buffer čeká na naplnění proměnné answer_buffer. Důležité je, že tento kód běží v jiném vlákně než metoda listen_in_thread. Takto počkáme, až metoda listen_in_thread proměnnou answer_buffer naplní a je navrácen text odpovědi. Potom metoda get_answer nastaví proměnnou not_listen na false (opět posloucháme příkazy) a vrátí text odpovědi, který pak může být zpracován.

Nyní je třeba vysvětlit, proč kód zpracovaní příkazu, při kterém je možné, že bude potřeba položit doplňující otázku uživateli, běží ve zvláštním vlákně. Problém je v tom, že o nutnosti otázky na uživatele se rozhoduje v jiné třídě. Pokud bych chtěl dostat odpověď v třídě Collection, musel bych v ní také začít poslouchat. Ale na poslouchání už máme třídu Listener a proto by vzhledem k čitelnosti kódu měla poslouchat jen tato třída. Navíc tu běží nekonečná smyčka, která čte vstup pojmenované roury a bylo by třeba ji zastavit a posléze pustit. Proto se poslouchání radši přenechá třídě Listener a aby se nekonečná smyčka mohla pro případný poslech odpovědi dostat ke čtení pojmenované roury, je třeba aby se vytvořilo nové vlákno, které dokončí zpracování příkazu, zatím co staré vlákno se může věnovat poslouchání.

Práce s proměnnými, které používají obě vlákna, tedy *not_listen* a *answer_buffer* , jsou uzavřeny do kritických sekcí. Navíc vlákno, které čeká na naplnění bufferu odpovědí, je uspáno do doby, než druhé vlákno odešle signál, že buffer je naplněn. Tato synchronizace vláken je implementována pomocí tříd *Mutex* a *ConditionVariable*.

4.3 Třída Collection

Třída Collection, jak už název napovídá, má na starosti práci s kolekcí hudby. Stará se především o přidávání skladeb do playlistu hudebního přehrávače Amarok. Třída Collection se nachází v souboru "collection.rb".

Při startu programu je vytvořen globální objekt collection. Při jeho vytvoření dochází k načtení celé kolekce do paměti pro rychlejší vyhledávání a pružnou reakci na požadavky uživatele.

Pro uchování dat v paměti je použita vlastní datová struktura. Jde o třídu MyHash, která je odvozená od třídy Hash. Hash je datová struktura podobná poli, ale index neboli klíč není nutně integer, ale může to být libovolný objekt. Oproti třídě Hash se MyHash liší tím, že pokud chceme přidat prvek s klíčem, který už Hash obsahuje, není původní prvek přepsán, ale je vytvořeno pole obsahující nový prvek a původní prvek, případně prvky, pokud již

předtím, bylo pod daným klíčem ukryto pole. Toto si můžeme dovolit díky dynamičnosti jazyka Ruby, kde proměnná mění svůj typ podle toho, co do ní ukládáme.

V této struktuře používáme jako klíče názvy skladeb, interpretů, alb nebo hudebních žánrů. Hodnotou je potom id záznamu v tabulce, která je součástí hudební kolekce. V programu je pak snadné získat id záznamu pomocí struktury MyHash, a podle něj pak najít v tabulce skladbu, nebo použít id alba, žánru či interpreta pro získání seznamu odpovídající skladeb, a pomocí URL uvedeného v těchto záznamech pak přidat skladbu do playlistu přehrávače Amarok.

Třída Collection obsahuje funkce pro načtení kolekce do paměti, přidání skladby do playlistu, přidání skladeb od určitého autora do playlistu, přidání skladeb určitého žánru do playlistu a přidání celého alba do playlistu. Tyto funkce jsou využity pro hlavní funkci této třídy, která se jmenuje add_to_playlist . Tato funkce zjistí jestli se v kolekci nachází skladba, album, interpret nebo žánr jména v proměnné name. Pokud ne, oznámí to, pomocí objektu speaker třídy Speaker. Pokud se v kolekci nachází jen jedna z možností, je zavolána patřičná funkce pro přidání skladby, interpreta, alba či žánru do playlistu. Pokud je více možností, je pomocí funkce what_to_play vygenerován seznam možností a opět pomocí objektu speaker, je uživatel dotázán, co si přeje přehrát. Podle odpovědi je poté zavolána patřičná funkce pro přidání do playlistu.

Funkce pro přidávání do playlistu add_album, add_genre, add_track a add_artist fungují víceméně podobně.

4.4 Třída Speaker

Třída Speaker, nacházející se v souboru "speaker.rb", se stará o hlasový výstup programu. Má jedinou funkci, která se jmenuje say a jako parametr má text. Tato funkce provádí volání funkce sayText programu kttsd pomocí protokolu DCOP. Funkce say při volání sayText vyplňuje dva parametry, první parametr je text určení k hlasové syntéze, tento parametr je vyplněn parametrem funkce say. Druhý parametr je jazyk, který se má k syntéze použít, ten je nastaven na angličtinu.

Jedinou proměnnou třídy Speaker je proměnná language. Ta je zde pouze proto, aby bylo v budoucnu snadné případné přepsaní programu do jiného jazyka.

Při spuštění programu se vytvoří globální objekt speaker, který se pak používá v celém programu pro komunikaci směrem k uživateli.

4.5 Třída Amarok a třída Track

Třída Amarok se nachází v souboru "amarok.rb". Je určena k ovládání přehrávače Amarok pomocí protokolu DCOP. Obsahuje metody pro nejběžnější funkce přehrávače, jako je zastavení a spuštění přehrávače, přechod na další nebo předchozí skladbu, práce s hlasitostí nebo vyčištění playlistu.

Každá funkce provádí jednoduché zavolání vzdálené funkce přehrávače Amarok. Existence této třídy má především napomoci lepší čitelnosti kódu a zabalení vzdáleného volání funkcí pod metody třídy Amarok.

Soubor "amarok.rb" obsahuje ještě třídu Track, které slouží k zjištění informací o aktuálně přehrávané skladbě. Tyto informace se zjistí, při vytvoření třídy. Zavoláním jediné metody třídy Track získáme text, určený k ohlášení informací o právě hrané skladbě uživateli.

4.6 Skript generate_gram

Jde o jednoduchý skript v jazyce Ruby, který přečte z kolekce přehrávače Amarok obsažené skladby, interprety, alba a žánry a sestaví z nich gramatiku pro rozpoznávací program recognition. Skript se nachází v souboru "generate_gram.rb" a na velké kolekci hudby může jeho běh trvat i několik minut.

Kapitola 5

Používání programu

V této kapitole popíši, jak program nainstalovat a jak ho spustit. Popíši jak se program ovládá, jaké akceptuje příkazy, jak na ně bude reagovat a situace, do kterých se můžeme při používání programu dostat.

V první části se věnuji přípravě na instalaci a samotné instalaci programu. V dalších částech pak samotnému ovládání programu. Nejprve popíši jednoduché příkazy v části nazvané Základní ovládání. Tyto příkazy se týkají především ovládání samotného přehrávání.

Část Pokročilejší ovládání se věnuje především příkazům, které pracují s kolekcí přehrávače. Použití těchto příkazů je pro uživatele složitější, protože příkazy nemusí být jednoznačné a program může vyžadovat po uživateli upřesnění požadavku.

Poslední část této kapitoly obsahuje doporučení pro používání programu, protože uživateli jistě nestačí jen program spouštět, ale využívat ho a to nejlépe, jak je to jen možné.

5.1 Instalace programu

Před instalací samotného programu je nutné nainstalovat do systému všechny závislosti. Předně je nutné ověřit si, že v systému máme nainstalovaný interpret jazyka Ruby. Dále je třeba mít v systému používané knihovny. Program Festival v systému většinou již je, stejně jako KTTS, je dobré se ale ujistit. Dále je nutné nainstalovat knihovnu pro rozpoznávání hlasu CMU Sphinx.

Samotná instalace programu je jednoduchá. Po stažení programu z internetu ve formě balíčku *ascon.amarokscript.tar.gz*, spustíme Amarok, v menu vybereme otevření správce skriptů a v něm zvolíme "Nainstalovat skript".

Potom vybereme balíček *ascon.amarokscript.tar.gz*. Amarok už si doplněk nainstaluje sám.

Poslední potřebnou věcí je nastavení části starající se o rozpoznávání řeči. Veškerou konfiguraci je možné provést spuštěním skriptu *install.sh* . Ten se uživatele zeptá na cestu ke knihovně CMU Sphinx a vše ostatní zařídí sám. Tato instalace zahrnuje vygenerování gramatiky z kolekce hudby. Uložení této gramatiky do archivu *Recognizer.jar* a jeho nakopírování do adresáře s knihovnou. To z důvodu, aby se uživatel nemusel starat o správné nastavení cest ke knihovnám.

Aby instalace proběhla pro uživatele co nejpříjemněji, spustí program instalační skript sám, pokud nenajde svůj konfigurační soubor. Díky tomu je instalace celého programu velmi přímočará, jednoduchá a tedy přesně taková, jakou si jí uživatelé přejí.

Pokud by chtěl uživatel použít vlastní program pro rozpoznávání řeči, stačí takový program nastavit, aby vypisoval text na standardní výstup a umístit příkaz pro spuštění příkazu do souboru *ascon.conf* a příkaz pro aktualizaci gramatiky do souboru *update.conf* .

5.2 Základní ovládání

Program umí devět základních příkazů, které přebírají základní funkce grafického rozhraní přehrávače Amarok. Jde o funkce, které nevyžadují žádnou interakci s uživatelem a neočekávají, že by došlo k nějaké chybě.

Zde je seznam základních příkazů s krátkým vysvětlením jejich funkce.

- **play** Spustí přehrávání nastaveného seznamu skladeb.
- **pause** Zastaví přehrávání skladby a čeká na aktuální pozici.
- **next** Začne přehrávat další skladbu, pokud další skladba neexistuje, zastaví přehrávání.
- **previous** Začne přehrávat předchozí skladbu, pokud předchozí skladba neexistuje, zastaví přehrávání.
- **stop** Zastaví přehrávání skladby a vrátí aktuální pozici přehrávání před tuto skladbu.
- **info** Informuje uživatele o aktuálně přehrávané skladbě. Hlasová informace je doprovozena textovou informací na obrazovce.

- **mute** Hlasitost přehrávání je nastavena na minimum, při opětovném zadání příkazu mute je hlasitost nastavena zpět na původní hodnotu.
- **volume up** Zvýší hlasitost přehrávání.
- **volume down** Sníží hlasitost přehrávání.

5.3 Pokročilejší ovládání

Pokročilejší příkazy už nejsou jen jednoduché fráze, ale v podstatě celé věty. Skládají se z pevně daného začátku a volitelné části, která by měla odpovídat názvu skladby, interpreta, alba nebo žánru z kolekce.

Tyto pokročilé příkazy pracují s obsahem hudební kolekce přehrávače Amarok. Příkazy nemusí být jednoznačné a proto může být potřeba interakce s uživatelem kvůli upřesnění jeho požadavku.

Požadovaná skladba, album, žánr nebo interpret nemusí být v kolekci přítomen a proto může vykonávání takového příkazu skončit chybou.

Prvním příkazem je příkaz *play me*. Pevná část je právě fráze *play me* po které může následovat název skladby, interpreta, alba nebo žánru. Pokud se takový název v kolekci nevyskytuje, program to uživateli oznámí, zpracování příkazu skončí a program čeká na další příkaz.

Pokud je z názvu jasné, jestli jde o skladbu, interpreta, album nebo žánr, tedy pokud se v kolekci nenachází například interpret a album stejného názvu, nebo více skladeb stejného názvu, naplní se seznam skladeb a spustí se přehrávání.

Pokud se v kolekci nachází více věcí stejného jména, není tedy jasné co by se mělo vybrat a proto se program zeptá uživatele. Sdělí uživateli očíslovaný seznam možností, kde si pomocí čísel přiřazených jednotlivým možnostem může vybrat co chce přehrávat. Nebo může uživatel říct *all* a do seznamu skladeb jsou přidány všechny možnosti.

Při výkonu tohoto příkazu je zastaveno přehrávání. Seznam skladeb je před přidáním odpovídajících skladeb vyčištěn a po jeho naplnění je spuštěno přehrávání.

Stejně s kolekcí pracuje i příkaz *add to playlist*. Po frázi *add to playlist* následuje název skladby, interpreta, alba či žánru. Pokud je volba nejednoznačná, je uživatel programem vyzván k upřesnění výběru, stejně jako při použití příkazu *play me*. Naopak oproti příkazu *play me* není zastaveno

přehrávání, skladby jsou přidány do seznamu skladeb, který před tím není vyčištěn a po přidání skladeb není spuštěno přehrávání.

5.4 Doporučení pro použití

Pokročilejší ovládání programu zmíněné v minulé části je jistě tím nejlepším, co program nabízí. Pokud ale chceme, aby vyhledávání v kolekci a přehrávání skladeb na požádání bylo přesné, musí být přesná i naše kolekce.

Program umí vyhledávat v kolekci podle jména skladby, interpreta, alba nebo žánru a počítá s tím, že se takové informace v kolekci nacházejí. Pokud informace o skladbě v kolekci není, tak takovou skladbu program nikdy nenajde a nikdy ji nepřehraje. Pokud tedy chybí informace o polovině skladeb, nebo nejsou přesné, pro program takové skladby, jako by ani neexistovali.

Pro plné využití tohoto programu a především funkcí pro práci s kolekcí je dobré, mít u všech skladeb v kolekci informace o názvu sklady, interpretovi, albu a žánru skladby. Tyto informace se dají do kolekce doplnit ručně v prostředí přehrávače Amarok, nebo programy pro editaci mp3 tagů, z nichž některé umí zjistit tyto informace automaticky z databáze na internetu.

Kapitola 6

Využití a další možný vývoj

Používání programu může mít vždy v podstatě dva důvody. Buď program používat chceme, proto ho používáme a až nebudeme chtít, bez žádných následků program přestaneme používat, nebo program musíme používat a bez něho o něco přicházíme. Tímto způsobem se dají charakterizovat i dvě možné skupiny uživatelů, které budou využívat tento program a přínos programu pro tyto skupiny bude probrán v této kapitole.

Kromě toho zkusíme předpovědět, jak se program bude vyvíjet v budoucnu. Již jsem zmínil, kolik problémů přináší nové prostředí KDE4, nabízejí se ale i další směry, jak práci a vědomosti získané prací na tomto projektu zúročit.

6.1 Využití pro běžné uživatele

Většina lidí si pod pojmem hlasové ovládání počítače představí rozhovor kapitána Spocka s palubním počítačem vesmírné lodi Enterprise. Převládá názor, že se hlasem komunikuje s jakousi umělou inteligencí, chytřejší než člověk. Proto většina lidí od ovládání hlasem očekává, že si s počítačem bude povídat. Po zjištění, že se pro komunikaci musí používat přesné příkazy nadšení opadá, z možnosti ovládání hlasem se stává jen super vlastnost, se kterou je dobré pochlubit se kamarádům, ale pro každodenní používání se nehodí. Přeci jen, než přeříkat příkaz pro přehrání určité skladby odpovídající nějakému požadavku a pak počítači vysvětlovat, kterou z variant jsme měli na mysli, je jednodušší daný požadavek napsat do políčka k tomu určenému a posléze kliknout na požadovanou skladbu.

Možnost každodenního použití klesá i s přesností rozpoznávání řeči. Chyby

v rozpoznávání se stávají otravnějšími a neustálé opravování počítače nás otravuje. Pokud je tedy uživatel přítomen u myši, klávesnice a monitoru, nemá ovládání hlasem pro uživatele v podstatě žádný přínos.

Hlasové ovládání může mít přínos v případě, že jsme od počítače dál, například v posteli. Pustit si kolekci hudby na dobrou noc a mít možnost počítač okřiknout, že tuto skladbu nemáme rádi a měl by přejít na další je jistě lákavá. Musíme si ale uvědomit, že pro toto potřebujeme mít ve své blízkosti mikrofon. To také ztěžuje využití programu. Na druhou stranu je toto využití asi nejreálnější, krátké příkazy jsou většinou rozpoznány dobře a mít u postele položený mikrofon není o nic náročnější, než tam mít položené dálkové ovládání.

6.2 Využití pro zrakově postižené

Během mé práce a hlavně pak při samotném testování mého programu jsem si uvědomil, že nejvíc ocení můj program zrakově postižení uživatelé. Ovládání počítače pomocí hlasu je pro ně ideální možnost, jak s počítačem pracovat. Můžou se ptát, zadávat příkazy a počítač jim odpovídá. Pro takové uživatele jen tento způsob komunikace zcela běžný.

Pokud uživatel vidí, raději si deset možností, které dostane na výběr od programu přečte, protože než by si poslech poslední, zapomněl by prvních pět. Ne však uživatel, který vidí špatně, nebo nevidí vůbec. Ten bude rád, že takovou možnost má. I proto byl program a jeho chování uzpůsobeno těmto lidem a tedy po své instalaci a zapnutí již nepotřebuje žádný visuální kontakt s uživatelem a vystačí si pouze s komunikací pomocí hlasu.

6.3 Další vývoj

Jak jsem již zmínil v předcházející kapitole, další vývoj hudebního přehrávače Amarok a prostředí KDE4 prozatím neumožňuje snadnou úpravu pro novou verzi přehrávače Amarok. Nicméně program je navržen tak, aby byl snadno modifikovatelný pro jiný přehrávač, stačí modifikovat třídu starající se o ovládání přehrávače. V budoucnu je tedy možné poměrně snadno program přepsat pro jiné hudební přehrávače, případně video přehrávače nebo jiné programy. Smysluplné využití si dovedu představit například s programem pro agregaci novinek, tzv. rss čtečky nebo i internetovým prohlížečem.

Kapitola 7

Závěrečné shrnutí

Výsledkem mé snahy je pro uživatele velmi snadno použitelný program, který naprosto mění zažité ovládání přehrávače Amarok. Odlišuje se i od ostatních hlasově ovládaných aplikací. Takové aplikace většinou rozpoznají hlasový příkaz, provedou ho, nebo vyskočí na uživatele nějaká hláška. Naproti tomu můj program s uživatelem opravdu komunikuje, na hlasový příkaz reaguje program opět hlasem, v tom je jedinečný.

Přes poměrně velkou závislost na externích knihovnách se podařilo instalaci programu udělat jednoduchou stejně, jako je tomu u ostatních doplňků pro přehrávač Amarok. I v ovládání se podařilo zachovat jednoduchost a zároveň velkou možnost výběru, co vlastně chce uživatel přehrát, a to při minimálním počtu příkazů.

Použitelnost programu sice trochu kazí nepříliš vysoká úspěšnost knihovny pro rozpoznávání řeči, ale lepší knihovna zatím bohužel dostupná není. Naštěstí se program podařilo navrhnout a implementovat tak, aby jeho části šli snadno nahradit jinými a tedy nebude problém využít lepší knihovnu pro rozpoznávání řeči, pokud se taková objeví. Toto rozdělení programu na několik minimálně závislých částí přináší i možnost snadné úpravy doplňku pro jiný přehrávač, nebo i jiný program, jak je popsáno v předešlé kapitole.

Během práce na programu jsem přečetl stovky stránek o problému syntézy a rozpoznávání řeči. Naučil jsem se programovací jazyk Ruby, psát program, který má více nezávislých částí, používá spoustu externích programů a má i několik nezávislých vstupů dat a musí proto běžet v několika vláknech. Při používání a testování programu jsem pochopil i uživatelské požadavky a zahrnul je do svého řešení.

Zakončením mé snahy je tato práce. Myslím, že se mi v podařilo čtenáři

předat své zkušenosti. Pro mírně zkušeného programátora by neměl být problém po přečtení této práce vytvořit si vlastní program ovládaný hlasem.

I když práce není návodem pro programátory, ani pro běžné uživatele, každý v ní může najít užitečné informace. V práci jsem zmínil základní popis programu pro vytvoření si představy o něm. Popsal jsem použité technologie jejich srovnání s alternativami. Také jsem popsal, jak je celý projekt řešen, rozdělen na menší programy, včetně nejdůležitějších tříd a metod. Nechybí ani popis interakce mezi programem a uživatelem, jejíž princip je jiný než v běžně používaných programech.

Na závěr bych rád vyjádřil přání, aby tento text nebyl jen zakončením práce na mém projektu, ale také začátkem nového vývoje v ovládání počítače. Doufám, že vývoj přinese přesnější rozpoznávání řeči a následovat budou další programy podobné tomuto. S vývojem umělé inteligence se snad dočkáme i programů mnohem lepších, než je tento.

Literatura

[1] Kolektiv autorů: *Ruby* , informace o programovacím jazyce Ruby, dostupné na internetu: *http://www.ruby-lang.org/*

[2] Kolektiv autorů: *CMU Sphinx: The Carnegie Mellon Sphinx Project* , informace o systému CMUSphinx, dostupné na internetu: *http://cmusphinx.sourceforge.net/*

[3] Clark R., Black A.: *The Festival Speech Synthesis System* , informace o systému Festival, dostupné na internetu: *http://www.cstr.ed.ac.uk/projects/festival/*

[4] Brawn P., Ettrich M.: *DCOP: Desktop COmmunication Protocol*, dokumentace k protokolu DCOP, dostupné na internetu: *http://developer.kde.org/documentation/other/dcop.html*