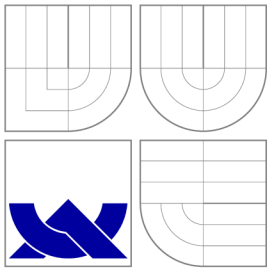
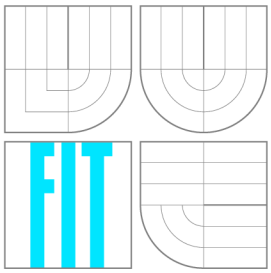


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

XMPP ROBOT S ROZHRANÍM V PŘIROZENÉM JAZYCE

XMPP ROBOT WITH NATURAL LANGUAGE INTERFACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAGDALÉNA KRYGIELOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MAREK SCHMIDT
SUPERVISOR

BRNO 2009

Abstrakt

V této práci je popsán XMPP robot komunikující v přirozeném jazyce. Jeho úkolem je vytvářet rozhraní pro různé služby, zejména pro vyhledávání vlakových a autobusových spojení. Tento robot umožňuje lidem používajícím síť Jabber získávat informace týkající se implementovaných služeb tak, jak je jim přirozené, bez nutnosti vyplňovat formuláře, nebo pamatování si formátu příkazu.

Abstract

This bachelors thesis describes an XMPP robot which communicates using natural language. It works as an interface for various services, especially for providing information about train and bus connections. This robot allows people using Jabber network to get the information about implemented services the way which is natural for them, without the need of filling forms or remembering command formats.

Klíčová slova

dialogový systém, přirozený jazyk, XMPP, jabber, Python, Jabbim

Keywords

dialog system, natural language, XMPP, jabber, Python, Jabbim

Citace

Magdaléna Krygielová: XMPP robot s rozhraním v přirozeném jazyce, bakalářská práce, Brno, FIT VUT v Brně, 2009

XMPP robot s rozhraním v přirozeném jazyce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Marka Schmidta. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Magdaléna Krygielová
26. května 2009

Poděkování

Na tomto místě bych chtěla poděkovat svému vedoucímu práce Ing. Marku Schmidtovi za vstřícný přístup a hodnotné připomínky při vedení bakalářské práce.

Obsah

Úvod

V současnosti existuje spousta různých služeb, které nějakým způsobem podávají jejich uživatelům informace. Různé vyhledávače všeho druhu, encyklopedie, slovníky, informace o dopravních spojeních pro nás nejsou žádnými novinkami. Vyžadují ale, aby jejich uživatele nejdříve spustili příslušnou aplikaci nebo webový prohlížeč a poté zadali svůj požadavek, v naprosté většině případů vyplněním nějakého formuláře.

V této práci se chci pokusit o to, aby k informacím poskytovaným těmito službami bylo možné přistupovat jinak. V první řadě tak, aby nebylo nutné zdlouhavě vyplňovat formuláře, nebo si pamatovat nějaký konkrétní formát příkazu, ale aby se uživatel mohl zeptat tak, jak je mu přirozené, tedy v přirozeném jazyce, konkrétně v češtině. V řadě druhé chci, aby tyto informace bylo možné získat na jednom místě, prostřednictvím jednoho robota, který by sám rozlišoval, který dotaz patří které službě.

V první kapitole se nachází stručný přehled teoretických znalostí, které je dobré mít při tvorbě tohoto druhu robota, místy je i zmínka o implementacích knihoven, které je vhodné použít. V kapitole druhé popisuji návrh robota a v kapitole třetí popisuji, jak je robot implementován. Kapitola čtvrtá se věnuje způsobu testování aplikace, ukládání rozhovorů a vyhodnocení testování na uživatelích. V závěru je shrnutí výsledku práce a jsou zde uvedeny možnosti dalšího vývoje aplikace.

Kapitola 1

Analýza

Na začátku této kapitoly se nachází formulace cíle a stručný nástin toho, jaké podobné systémy již existují. Dále je zde přehled teoretických znalostí, které je vhodné mít při tvorbě dialogového systému komunikujícího v přirozeném jazyce, a jsou zde také uvedeny implementace knihoven, které je vhodné použít.

1.1 Formulace cíle

Cílem této práce bylo navrhnout a implementovat XMPP[1] robota jako rozhraní nad službami serveru Jabbim[2] umožňujícího komunikaci v přirozeném jazyce. Mezi tyto služby patří například služba vyhledávání vlakových a autobusových spojení, cizojazyčný slovník, informace o svátcích... Tyto služby nabízí jiný robot a jsou uživatelům sítě jabber běžně dostupné, ale tento robot vyžaduje, aby zprávy s dotazy na jeho služby byly v určitém formátu a pamatování si formátu dotazu činí většině lidí problémy.

Úkolem je tedy vytvořit robota, který by přijímal požadavky zapsané v přirozeném jazyce a tyto požadavky překládal do formátu zpráv, které přijímá robot serveru Jabbim.

Robot by měl tvořit rozhraní primárně pro službu vyhledávání vlakových a autobusových spojení.

1.2 Existující implementace

Jedním z prvních systémů komunikujících s člověkem v přirozeném jazyce byl program ELIZA. Byl vytvořen v letech 1964–1966 Josephem Weizenbaumem a celkem úspěšně imitoval rozhovor s psychoterapeutem. [2] Tento robot byl inspirací pro vytvoření dalších podobných systémů, jako například v dnešní době celkem úspěšného robota A.L.I.C.E.[3] Slušný přehled konverzačních robotů lze nalézt na WWW stránkách Chatbots.org[4].

Tyto roboty slouží primárně dvěma účelům — buď pro pobavení a zabití času lidí, kteří s nimi konverzují, nebo poskytují nějaké užitečné informace na konkrétní témata. Tyto poslední bývají často umístěny na webových stránkách různých firem a kromě informací o záležitostech, kterými se firma zabývá, také navigují uživatele po těchto stránkách. Jako příklad uvedu robotku Annu fungující na webových stránkách společnosti IKEA[5], která,

[1] http://xmpp.org/
[2] http://www.jabbim.cz/
[3] http://alice.pandorabots.com/
[4] http://www.chatbots.org/
[5] http://www.ikea.com/cz/

jako jeden z mála takových robotů, dovede komunikovat v českém jazyce.

Předností mnou vytvořeného systému by mělo být to, že bude s uživatelem komunikovat v českém jazyce a poskytovat užitečné informace.

1.3 XMPP

Robot má komunikovat pomocí protokolu XMPP (Extensible Messaging and Presence Protocol). XMPP je založený na značkovacím jazyce XML. Vznikl jako protokol pro síť Jabber a byl zaměřen na posílání zpráv a zjišťování informace o přítomnosti. Jednou z jeho předních vlastností je otevřenost. Znamená to, že specifikace protokolu jsou všem dostupné a software komunikující protokolem XMPP může implementovat každý, kdo o to má zájem. Mezi další výhody XMPP protokolu patří decentralizace, bezpečnost, flexibilita. [3]

1.3.1 XMPP knihovny

Pro jazyk Python je k dispozici několik knihoven implementujících komunikaci pomocí protokolu XMPP. Zmíním jen Twisted Words[6], xmpppy[7], jabber.py[8].

Twisted

Twisted je framework pro síťové programování podporující mnoho protokolů. Je napsaný v jazyce Python. Je tvořen mnoha podprojekty, mezi které patří také projekt Twisted Words poskytující funkce pro tvorbu IM[9] serverů a klientů. V robotovi použiji právě tento framework, protože je aktivně vyvíjen a snadno se jej používá.

1.4 Reprezentace znalostí

Jednou z věcí, kterou je třeba se zabývat, je způsob reprezentace znalostí, který definuje, jak mají být získané informace uchovány a používány.

K reprezentaci znalostí jsou nejvíce využívány:

- logické modely
- síťové modely
- produkční systémy

1.4.1 Logické modely

V logických modelech jsou znalosti vyjádřeny formulemi některé logiky – nejčastěji predikátové logiky prvního řádu. Tento způsob reprezentace je užitečný pro uchování jednoduchých faktů. Díky odvozovacím pravidlům je možné na základě stávajících znalostí odvozovat nové znalosti, nebo provádět kontrolu, zda některé formule nejsou ve vzájemném protikladu [6, s. 658].

[6] http://twistedmatrix.com/trac/
[7] http://xmpppy.sourceforge.net/
[8] http://jabberpy.sourceforge.net/
[9] instant messaging

1.4.2 Síťové modely

Síťové modely slouží k reprezentaci objektů i jejich vzájemných vztahů. K nejznámějším patří:

- sémantické sítě
- sémantické rámce
- scénáře

Sémantické sítě

V sémantických sítích jsou znalosti reprezentovány formou sítě uzlů spojených hranami. Uzly představují jednotlivé objekty nebo pojmy a hrany reprezentují vztahy mezi nimi. Jsou graficky reprezentovány orientovaným grafem.

K výhodam reprezentace znalostí pomocí sémantických sítí patří snadné přidávání a modifikace znalostí (v porovnání s logickými modely), sémantické sdružování příbuzných znalostí a jejich hierarchické členění. Další důležitou vlastností těchto sítí je možnost reprezentace dědičnosti. Nevýhodou je obtížná manipulace s uloženými znalostmi.[6, s. 661], [10]

Sémantické rámce

Sémantické rámce mohou být považovány za rozšíření sémantických sítí. Narozdíl od sémantických sítí, je zde objekt a s ním svázané vlastnosti soustředěn do jednoho rámce. Vlastnosti objektu jsou v rámci reprezentovány *sloty*. Hodnoty slotů se nemusejí určovat při vytváření rámce, ale mohou se doplňovat postupně, v závislosti na aktuálním množství vlastností, které o objektu víme. Mohou také zůstat nevyplněny. Sloty mohou obsahovat konkrétní hodnoty vlastností, nebo odkazy na další rámce. Tím se vytváří hierarchická struktura rámců, která má, jako sémantické sítě, vlastnost dědičnosti. Mohou být také používány k zapamatování procedur, které s hodnotami slotů rámce manipulují. [6, s. 662]

Scénáře

Scénář je struktura pro popis posloupnosti událostí, kde jedna zapříčiní vznik té následující. Tato struktura je podobná rámci, protože také obsahuje sloty. [6, s. 663]

1.5 Dialogový systém

Dialogový systém je systém určený ke konverzaci s člověkem. Tyto systémy využívají různé formy komunikace, jako text, řeč, obraz, hmat... [1] Náš robot bude ve své podstatě dialogový systém a s člověkem bude komunikovat textovou formou.

1.5.1 Dialogový manažer

Jádrem dialogového systému je dialogový manažer. Jeho úkolem je přijímat zprávy od uživatele, rozhodovat o odpovědi pro uživatele, získávat data z jiných zdrojů, udržovat historii dialogu... Dialogový manažer musí všechno řídit tak, aby požadavky uživatele byly uspokojeny. [6, s. 670]

1.5.2 Strategie řízení dialogu

Dialogový manažer používá určitou strategii řízení dialogu. Rozlišujeme tři základní typy dialogových strategií [6, s. 674]:

- s iniciativou systému
- se smíšenou iniciativou
- s iniciativou uživatele

Strategie s iniciativou systému

V dialogové strategii s iniciativou systému je řízení celého dialogu zajišťováno systémem. V systému s touto strategií nemá uživatel možnost ovlivnit směr, kterým se dialog ubírá, pouze odpovídá na otázky, které mu systém pokládá. Často musí svoje odpovědi vybírat z několika předem definovaných odpovědí, které mu systém nabízí formou nápovědy. [6, s. 674]

Strategie se smíšenou iniciativou

V dialogové strategii se smíšenou iniciativou je uživateli dána větší volnost v komunikaci se systémem. Systém se stará o to, aby byl dosažen celkový cíl dialogu (uspokojit předpokládaný požadavek uživatele), ale uživatel má větší volnost ve formování odpovědí na dotazy systému. [6, s. 675]

Strategie s iniciativou uživatele

V dialogové strategii s iniciativou uživatele je každý uživatelův požadavek interpretován bez předem daných omezení. V těchto systémech může během dialogu docházet ke změně tématu hovoru. Hlavně díky této vlastnosti se mi tato strategie jeví jako nejvíce vhodná pro našeho robota, protože ten bude tvořit rozhraní pro více služeb a uživateli by měla být dána možnost mezi těmito službami jednoduše přepínat. [6, s. 676]

1.5.3 Typy dialogových systémů

Podle toho, jak dialogový manažer zpracovává uživatelovy zprávy, můžeme jeho návrh klasifikovat do těchto skupin [6, s. 676–682]:

- systém s konečným počtem stavů
- systém využívající strukturu rámců
- systém založený na agentech

Dialogový systém s konečným počtem stavů

Dialogový systém s konečným počtem stavů využívá k reprezentaci struktury dialogu stavově přechodovou síť. Uzly této sítě reprezentují stavy dialogu – v těchto stavech systém očekává vstup od uživatele. Přechody mezi uzly určují možné cesty sítí a tím i všechny proveditelné dialogy.

V těchto systémech je většinou využívána dialogová strategie řízení s iniciativou systému. Je pro ně typická navigace formou jednoduchých nápověd (slova, fráze), ze kterých má uživatel vybírat odpověď.

Výhodou těchto systémů je jednoduchost dialogu. Možných odpovědí v každém stavu bývá omezený počet a většinou jsou nabízeny formou nápověd. Minimalizuje se tak riziko nesprávného pochopení uživatelovy odpovědi.

Nevýhodou je jejich malá pružnost, je nemožné uskutečnit dialogovou výměnu mimo již navrženou síť. Také ve většině případů není uživateli dovoleno zadat více položek najednou a dosažení cíle dialogu je proto pomalejší.

Dialogový systém s konečným počtem stavů je vhodný pro dobře strukturované úlohy, ve kterých v každém uzlu stavově přechodové sítě je spíše menší počet možností dalšího pokračování dialogu. [6, s. 676–678]

Systém využívající strukturu rámců

V tomto dialogovém systému je využíván rámec, nebo struktura rámců (viz. 1.4.2), které nějakým způsobem reprezentují řešenou úlohu. Aby mohla být provedena požadovaná akce, musejí být vyplněny požadované sloty rámce.

Takovýto dialogový systém v praxi funguje tak, že klade uživateli otázky, ve kterých se dotazuje na hodnoty slotů, které ještě nejsou vyplněné, a na základě odpovědí uživatele na tyto otázky doplňuje příslušné sloty. Uživatel může vkládat informace v různém pořadí, může také vkládat více informací najednou.

Složitější úlohy někdy potřebují využít systém několika rámců, obzvláště pokud má dialogový systém poskytovat služby ve více oblastech.

Hlavní výhodou tohoto typu dialogových systémů je možnost zadat více informací najednou, co může podstatně urychlit zadávání dotazu (oproti systémům s konečným počtem stavů) a také to, že nezáleží na pořadí zadávání informací. [6, s. 678–681]

Zejména díky těmto vlastnostem využijeme tohoto způsobu implementace dialogového systému i v našem robotovi.

Systém založený na agentech

Tento dialogový systém je postaven na architektuře se spolupracujícími agenty. Každý agent je navržen pro jednu konkrétní oblast a sdílí potřebná data s ostatními agenty. Využití této architektury je užitečné zejména pokud potřebujeme, aby systém vyřešil danou úlohu komplexně, ve více oblastech a s použitím více zdrojů informací. [6, s. 681–682]

Našeho robota by bylo možné také navrhnout jako systém založený na agentech, ale myslím si, že pro účely bakalářské práce je toto řešení příliš náročné.

1.6 Gramatika

Gramatika v informatice je prostředkem pro popis formálního jazyka. Je tvořena abecedou terminálních symbolů (terminálů), abecedou neterminálních symbolů (neterminálů), počátečním symbolem a pravidly. Abeceda je neprázdná konečná množina. Pravidla předepisují, jak lze symboly skládat do slov jazyka. Jazyk označuje množinu slov (řetězců) nad určitou abecedou. Neterminály jsou symboly, které se pomocí pravidel přepisují na terminály, které tvoří daný jazyk.

Podle Chomského hierarchie můžeme gramatiky rozdělit do 4 skupin na:

- gramatiky typu 0
- gramatiky typu 1 (kontextové gramatiky)
- gramatiky typu 2 (bezkontextové gramatiky)
- gramatiky typu 3 (regulární gramatiky)

Tyto gramatiky se liší tvarem pravidel. [7] [6, s. 653]

1.6.1 Bezkontextová gramatika

V bezkontextové gramatice jsou všechna pravidla tvaru $A \rightarrow w$, kde A je neterminál a w je řetězec, který může být tvořen jak terminály, tak i neterminály.[7]

1.6.2 Regulární gramatika

V regulární gramatice je každé pravidlo buď tvaru $A \rightarrow xB$ nebo tvaru $A \rightarrow x$, kde A a B jsou neterminály a x je řetězec neterminálů.[7]

Konečný automat

Regulární jazyky, tj. jazyky generované regulárními gramatikami, jsou rozpoznatelné konečným automatem.

Konečný automat má konečnou množinu stavů, ve které je specifikován počáteční stav a množina koncových stavů. Jeho součástí je také vstupní abeceda a konečná množina pravidel přechodů mezi stavy.

Na počátku se automat nachází v počátečním stavu. Dále v každém kroku přečte jeden symbol ze vstupní abecedy a přejde do stavu určeného pravidlem. Poté pokračuje čtením dalšího symbolu ze vstupní abecedy, dalším přechodem atd. Podle toho, zda automat skončí po přečtení vstupu ve stavu, který patří do množiny koncových stavů, platí, že automat buď daný vstup přijal, nebo nepřijal. [4]

Regulární výrazy

Regulární jazyky mohou být popsány rovněž regulárními výrazy.

Regulární výrazy se nejčastěji používají ve skriptovacích jazycích pro vyhledávání a úpravu textu. Obvykle se používají rozšířené definice regulárních jazyků umožňující jednodušší zápis běžných konstrukcí. Mezi nejznámější používané syntaxe patří *POSIX Regular Expressions* a *Perl Compatible Regular Expressions (PCRE)*. PCRE se používá mimo jiné i v jazyce Python. Tyto syntaxe obsahují prostředky, kterými lze popsat i silnější jazyky, než regulární. [5]

1.7 Morfologický analýzátor

Protože budeme pracovat s přirozeným jazykem, jistě se nám hodí morfologický analyzátor. Je to nástroj, který provádí lemmatizaci, tj. analyzuje slova a určuje jejich základní tvar (lemma) a gramatické kategorie.

Pro tento projekt je morfologický analyzátor velkou pomocí, protože při komunikaci s uživatelem robot často získává potřebná data v jiném, než základním tvaru, který potřebuje pro další zpracování. Bez něj by robot musel přijímat data pouze v základním tvaru a rozhovor by už nepůsobil příliš přirozeně.

1.7.1 Morfologické analyzátory češtiny

Jelikož náš robot má komunikovat v českém jazyce, je nutné použít morfologický analyzátor češtiny. Zmíním tyto projekty:

Morfologický analyzátor češtiny Ajka

Tento morfologický analyzátor vznikl v roce 1999 na Fakultě informatiky MU[10] v rámci diplomové práce Radka Sedláčka [8]. Mezi jeho funkce patří určení základního tvaru slova a jeho morfologických kategorií, generování možných tvarů zadaného slova, doplňování diakritiky a segmentace. Je vytvořena také knihovna *alib* pro použití Ajky v programech a rozhraní pro jazyk Python pro tuto knihovnu.

Morfologický analyzátor pro češtinu

Tento morfologický analyzátor byl vytvořen Stanislavem Černým [11] v roce 2008 na Fakultě informačních technologií VUT[11] v Brně. Stejně jako Ajka umí určit základní tvar slova a jeho morfologické kategorie a generovat možné tvary slova. Umí také zjistit číselnou hodnotu slova. Využívá knihovnu *libma* a pro tuto knihovnu je rovněž vytvořeno rozhraní pro jazyk Python. Všechny tyto nástroje jsou šířeny pod licencí GPL.

[10] Masarykova univerzita
[11] Vysoké učení technické

Kapitola 2

Návrh aplikace

V této kapitole bude popsán návrh XMPP robota a gramatických pravidel, které používá. Bude zde rovněž popsán způsob využití morfologického analyzátoru, komunikace se službami a vytváření odpovědi pro uživatele.

2.1 Základní návrh aplikace

Náš XMPP robot, kterého budeme vytvářet, je v podstatě dialogový systém. Jeho úkolem je tvořit rozhraní v přirozeném jazyce pro služby. Musí tedy rozumět přinejmenším tomu, že uživatel žádá o jednu z těchto služeb, vyhodnotit, zda mu uživatel poskytl všechny potřebné informace a pokud ne, tak se na ně uživatele zeptat. Musí samozřejmě také umět pochopit a zpracovat odpovědi na tyto otázky. Jakmile zná všechny potřebné informace, vytvoří dotaz na službu a odpověď pošle uživateli.

Aby dialog vypadal pokud možno přirozeně, je vhodné zvolit strategii řízení dialogu se smíšenou iniciativou nebo s iniciativou uživatele (viz 1.5.2). Tyto strategie dávají uživateli větší volnost ve vyjadřování jeho požadavků. To ale klade větší nároky na porozumění uživatelovy zprávy.

2.2 Návrh gramatiky

Porozumět uživatelově zprávě může pomoci gramatika. Vytvoříme soubor pravidel, která budou zahrnovat věty, nebo části vět, které od uživatele očekáváme. To, jaké věty od uživatele očekáváme, je úzce spjato se službami, které robot poskytuje. Čím více služeb robot poskytuje, tím větší rozsah možných vět uživatele se musí rozpoznat a to má samozřejmě vliv na množství gramatických pravidel. Na to, jak rozsáhlá bude gramatika, má vliv také to, zda potřebujeme rozumět celé větě, nebo si vystačíme jen s částí, ve které se vyskytují potřebné informace. Druhý způsob je mnohem méně náročnější na tvorbu pravidel a velikost gramatiky.

Požadavky na vytvořenou gramatiku jsou:

1. snadný zápis pravidel (kvůli přidávání dalších pravidel komunitou)
2. možnost určení morfologických kategorií vybraných frází

V případě našeho robota si vystačíme s bezkontextovou gramatikou, do které naroubujeme značky sloužící ke kontrole morfologickým analyzátorem.

Dále můžeme využít regulárních výrazů. Tím, že řetězce tvořené terminálními symboly bezkontextové gramatiky budou regulární výrazy a ne přímo slova přirozeného jazyka, se nám zápis gramatických pravidel velmi výrazně zjednoduší.

Výsledná pravidla mohou být zapsána například takto:

```
<den>       => dneska|zítra|dnes|(\d{1,2}\.\d{1,2}\.(\d{2,4})?)
<hodina>    => (\d{1,2}([:.]\d{1,2})?|(půl)? <s:k4>)
<hodina_p>  => <?predlozka> <!hodina> (h\.?|hodinu?)?
<cas>       => <hodina_p> <dennidoba_p>
<kdy>       => <!den> <cas>
```

Na levé straně je možné vidět neterminály a na pravé straně řetězce, na které se daný neterminál přepisuje. Tyto řetězce obsahují jak neterminály (symboly uzavřené v ostrých závorkách), tak terminály (symboly mimo ostré závorky).

Některé výše uvedené neterminály obsahují na začátku svého názvu symboly `?` nebo `!`.

Symbol `?` znamená, že je daný neterminál v tomto případě nepovinný. Tohoto výsledku by sice bylo možné dosáhnout i jinými způsoby, třeba vytvořením dalšího neterminálu, který by se přepisoval kromě určeného řetězce i na prázdný řetězec, ale toto řešení mi přijde elegantnější.

Symbolem `!` jsou označeny neterminály, kterých hodnotu si robot potřebuje zapamatovat.

V příkladu výše se vyskytuje ještě jeden neobvyklý zápis neterminálu. Jedná se konkrétně o `<s:k4>`, což je obyčejný neterminál `<s>`, který má ale navíc k sobě připojenou značku pro zpracování morfologickým analyzátorem, které si blíže popíšeme v sekci 2.5.

2.3 Pravidla přechodů

Další věcí, kterou je v návrhu třeba řešit, je schopnost určit, kterou službu chce uživatel v dané chvíli využít. Můžeme to zařídit třeba tak, že vytvoříme soubor pravidel, která nám tuto službu budou určovat.

Tento princip můžeme velmi dobře reprezentovat konečným automatem, kde stavy tohoto automatu představují jednotlivé služby a pravidla přechodů mezi stavy představují pravidla pro změnu právě využívané služby.

Prvky vstupní abecedy automatu jsou v tomto případě věty zapsané pomocí gramatických pravidel popsaných výše a společně tvoří všechny uživatelovy vstupy, které je robot schopen zpracovat.

Kvůli jednoduššímu přidávání těchto pravidel je vhodné mít tato pravidla zapsána v externím souboru. Pro větší přehlednost budou pravidla uložena ve více souborech, pro každý stav jeden.

Zde je malá ukázka možného zápisu pravidel pro koncový stav *spoj*:

```
hello: <!druh_spoje> <?kdy_odkud_kam>
spoj: z <!odkud:k1c2> <?kdy>
spoj: jsem v <!odkud:k1c6>
spoj: do <!kam:k1c2>
```

Část před dvojtečkou označuje aktuální stav, část za dvojtečkou je vstup pro toto pravidlo. Stav, do kterého se přejde po použití pravidla je určen názvem souboru.

2.4 Rámce

Jak jsem už zmínila výše, každý stav reprezentuje službu, kterou robot poskytuje. Výjimkou je jen počáteční stav, kdy se očekává zahájení konverzace ze strany uživatele, a robot zrovna žádnou službu neposkytuje.

Dále, pokud má robot poskytovat nějaké služby, je nutné, aby si pamatoval údaje, které mu o využití té služby uživatel poskytl. Pro tento účel se nabízí využití struktury rámců (viz. 1.4.2). Pro každý stav vytvoříme rámec a do jeho slotů budeme doplňovat údaje týkající se té dané služby. Údaje můžeme do rámce doplňovat postupně a je jedno, v jakém pořadí.

2.5 Morfologická analýza

V návrhu gramatiky jsem se zmínila o tom, že některé neterminály nesou také informaci o morfologických kategoriích dané fráze. Toto určení morfologických kategorií slouží jednak k omezení možných frází, které jsou pro daný úsek platné (např. chceme, aby se na nějakém místě mohla vyskytovat pouze číslovka), ale také pomáhají určit správný základní tvar slova (v případě homonym).

Převod slov na základní tvar i další funkce poskytuje knihovna morfologického analyzátoru. Použijeme knihovnu *libma* Stanislava Černého (viz 1.7.1), protože umí vše, co potřebujeme: analyzovat zadané slovo a určit jeho morfologické kategorie, ze základního tvaru a gramatických kategorií vytvořit slovo odvozené a umí určovat numerickou hodnotu číslovek, což je velmi výhodné např. při rozpoznávání zadaného času odjezdu vlaku.

Kromě rozpoznávání hodnot číslovek jsou funkce morfologického analyzátoru užitečné pro získávání základního tvaru měst a obcí při vyhledávání spojení a také při generování doplňujících otázek, kdy tyto názvy měst potřebujeme převést do tvaru v určitém pádu.

Formát neterminálu doplněn o údaje pro morfologickou analýzu:

<název neterminálu : značka : základní tvar>

Pod pojmem *značka* rozumíme zápis morfologických kategorií slova. Část obsahující základní tvar se používá, pokud chceme neterminál omezit na konkrétní slovo a slova od něj odvozená. V tomto případě uvedeme základní tvar tohoto slova a značku použijeme pro určení morfologických kategorií možných odvozených slov.

2.5.1 Značky

Morfologické kategorie analyzovaných slov budeme zapisovat pomocí značek. Tyto značky používá námi využívaná knihovna libma a jsou převzaty z analyzátoru Ajka [9]. Značka je tvořena dvojicemi znaků, kde první znak z dvojice určuje morfologickou kategorii a druhý hodnotu této kategorie.

Nejčastěji budeme používat tyto čtyři kategorie:

k – označuje slovní druh, *k1* tak znamená podstatné jméno, *k2* přídavné jméno, *k3* zájmeno, *k4* číslovku, *k5* sloveso, *k6* příslovce, *k7* předložku, *k8* spojku, *k9* částici a *k0* citoslovce

c – označuje pád, čili první pád bude zapsán jako *c1*, druhý *c2*, třetí *c3*, čtvrtý *c4* atd.

g – označuje rod, konkrétně pak *gM* označuje rod mužský životný, *gI* rod mužský neživotný, *gN* rod střední a *gF* rod ženský

n – označuje číslo, *nS* je číslo jednotné a *nP* je číslo množné

Tento popis není úplný, kompletní přehled značek lze nalézt na WWW stránkách Ajky v souboru tags.pdf [9].

2.5.2 Skloňování víceslovných názvů měst a obcí

V této části se chci podrobněji zabývat skloňováním názvů měst. Obzvláště těch, kterých název je tvořen více slovy. Toto skloňování měst je potřebné kvůli funkci hledání dopravních spojení.

Názvy měst tvořené dvěma slovy

Nejdříve se zaměřím na názvy měst a obcí o dvou slovech. Podle slovních druhů, kterými jsou tvořeny, a jejich pořadí, uvažuji tyto čtyři kategorie:

i. první slovo je přídavné jméno, druhé podstatné jméno (Český Těšín, Karlovy Vary)

ii. první je podstatné jméno, druhé přídavné jméno (Hradec Králové)

iii. obě podstatná jména, ale oddělená pomlčkou (Frýdek-Místek)

iv. obě podstatná jména (Město Albrechtice)

Každá z těchto kategorií se skloňuje jinak.

Do prvních dvou kategorií patří fráze, které obsahují jedno hlavní slovo, kterým je podstatné jméno. Při skloňování této fráze se skloňují obě slova, ale hlavní slovo určuje rod i číslo druhého slova. V první kategorii je hlavním slovem druhé slovo v pořadí, v druhé kategorii je jím první slovo.

Do třetí kategorie zařazuji názvy měst a obcí, kde jsou si obě složky rovnocenné a oddělují se spojovací čárkou. Tyto složky se skloňují samostatně a jsou na sobě nezávislé.

Do čtvrté kategorie zařazuji názvy jako Město Albrechtice, Město Libavá, Lázně Bělohrad. Složky těchto názvů jsou na sobě rovněž nezávislé, ale nabízí se zde otázka, zda se skloňují obě slova, nebo jen první, tedy pouze Město nebo Lázně. Nepovedlo se mi nalézt zdroje pojednávající o tom, jak tyto názvy skloňovat, ale na oficiálních stránkách těchto měst se setkávám jak s tvary, kdy je skloněna jen první část názvu (v Městě Albrechtice), tak i s tvary, kdy jsou skloněny obě části (v Městě Albrechticích), dovolím si tedy předpokládat, že jsou přípustné obě varianty.

Názvy měst tvořené třemi a více slovy

V naprosté většině tří a víceslovných názvů měst a obcí České republiky se vyskytuje předložka. Zpravidla jako druhé, nebo třetí slovo v pořadí.

Pokud je předložka druhým slovem v názvu, tak se před ní vyskytuje jednoslovný název města, nebo obce, který se skloňuje běžným způsobem.

Pokud je předložkou až třetí slovo, vyskytuje se před ní dvouslovný název spadající do jedné z kategorií popsaných výše a podle této kategorie se skloňuje.

Pád části vyskytující se za předložkou je dán konkrétní předložkou a pád celého názvu na ni nemá vliv. Takže při skloňování celého názvu na tuto část nemusíme brát ohled, protože je pro daný název neměnná.

Tuto vlastnost ukáži názorně na příkladě obce Nová Ves u Nového Města na Moravě. Je zde dobře vidět, že část před (první) předložkou se skloňuje, ale část za ní se nemění.

1. Nová Ves u Nového Města na Moravě
2. Nové Vsi u Nového Města na Moravě
3. Nové Vsi u Nového Města na Moravě
4. Novou Ves u Nového Města na Moravě
5. Nová Vsi u Nového Města na Moravě
6. Nové Vsi u Nového Města na Moravě
7. Novou Vsí u Nového Města na Moravě

2.6 Způsob komunikace se službami

V této části chci popsat, jakým způsobem by robot měl komunikovat se službami serveru Jabbim. Na tomto serveru funguje robot, který už má různé služby implementovány a náš robot bude v podstatě sloužit jako rozhraní v přirozeném jazyce k tomuto robotovi. Přezdívá se mu *Taxator* a je spuštěn s JID `netlabbot@njs.netlab.cz`.

Z nabízených služeb se zaměříme hlavně na vyhledávání spojů. Další službou, pro kterou bude náš robot tvořit rozhraní, je cizojazyčný slovník.

Taxator přijímá dotazy ve formátu:

`!`*`zkratka_služby parametry`*,

a po přijetí zprávy během chvilky odešle zpět odpověď.

Konkrétně pro službu slovníku by dotaz mohl vypadat například takto:

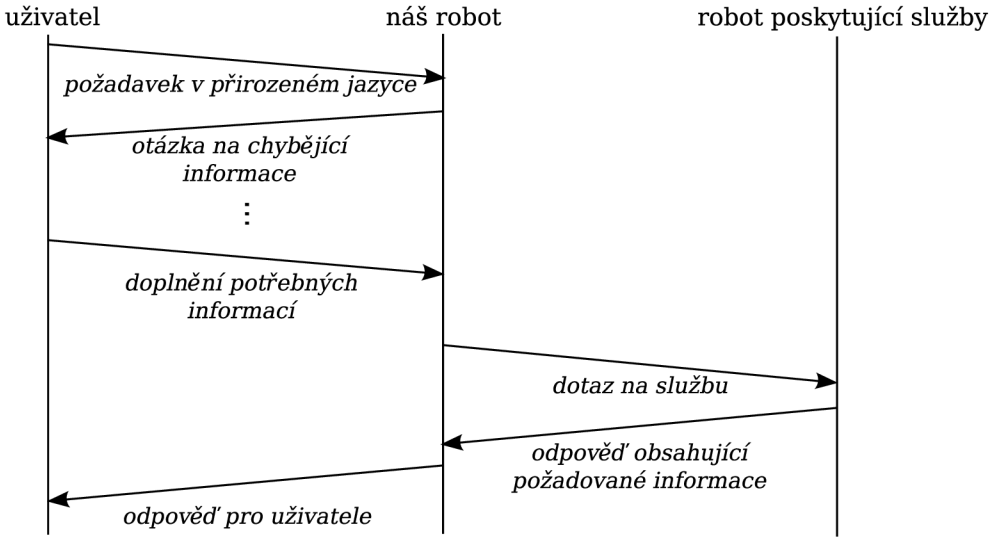
```
!slovnik encs orange,
```

a Taxatorova odpověď na tento dotaz by byla následující:

```
orange - pomeranč
orange - oranžový
orange - pomerančový
orange - oranž
orange - oranžová
orange - orančová
orange fin - mladý pstruh
orange juice - pomerančový džus
orange juice - džus
orange peel - pomerančová kůra
orange tree - pomerančovník
orange-yellow - oranžově žlutý
orangeade - oranžáda
orangery - skleník (podst.jm.) g
oranges - pomeranče
```

Průběh komunikace s Taxatorem je ukázán na obrázku 2.1. Stejným způsobem může ale robot komunikovat s jakýmkoliv jiným robotem poskytujícím služby prostřednictvím XMPP zpráv.

Prvně vyšle uživatel zprávu našemu robotovi a ten následně vyhodnotí, zda získal všechny informace, které potřebuje pro splnění požadavku. Pokud je nemá, začne se uživatele vyptávat na chybějící informace. Jakmile získá všechny potřebné informace, vytvoří z nich dotaz pro Taxatora a pošle mu zprávu s tímto dotazem. Poté počká na jeho odpověď, upraví ji do vhodného formátu a odešle uživateli.



Obrázek 2.1: Ukázka komunikace našeho robota s robotem poskytujícím služby.

2.7 Vytváření odpovědí pro uživatele

Náš robot, kromě toho, že dokáže zpracovat vstup uživatele v přirozeném jazyce, by měl také v přirozeném jazyce uživateli odpovídat.

Odpovědi robota je možné rozdělit do tří skupin.

2.7.1 Odpovědi bez předchozích znalostí

První skupinu tvoří jednoduché odpovědi na uživatelovy fráze, na které je možné odpovědět bez dalších znalostí. Patří tady hlavně povídací věty typu „není zač“ jako odpověď na poděkování, nebo „ahoj, co bys rád?“ jako odpověď na pozdrav.

Tyto odpovědi bezprostředně reagují na uživatelovy vstupy, je proto vhodné je propojit s pravidly přechodů.

Formát pravidel ukázaný v podkapitole 2.3 tedy můžeme upravit do této podoby:

```
hello: na něco zeptat >> jasně, ptej se
hello: <pozdrav> >> ahoj, co bys rád?
hello: <dik> >> není zač
```

V tomto zápisu jsou odpovědi vymezeny dvojicí znaků „>>“ a koncem řádku.

Aby byl rozhovor přirozenější, je dobré, aby robot neodpovídal pořád stejně, ale svoje odpovědi třeba náhodně vybíral ze sady podobných vět.

Pravidla by pak mohla vypadat takto:

```
hello: na něco zeptat >> jasně, ptej se | klidně, co chceš vědět?
hello: <pozdrav> >> ahoj, co bys rád? | ahoj | zdar!
hello: <dik> >> není zač
```

Různé varianty odpovědi jsou ohraničeny znakem „|“.

2.7.2 Doplňující otázky

Do druhé skupiny zařazuji odpovědi vyptávající se na údaje potřebné k vytvoření dotazu na službu. Použijí se ve fázi, kdy dialogový manažer ví, kterou službu chce uživatel využít, ale ještě nemá všechny potřebné informace. Tyto odpovědi můžeme zajistit tak, že vytvoříme soubor všech otázek, které můžeme potřebovat a z nich budeme vybírat na základě toho, který rámec je aktivní a toho, které sloty tohoto rámce jsou ještě prázdné a je třeba je doplnit.

Z toho důvodu je třeba tyto otázky napojit na konkrétní sloty jednotlivých rámců.

Je také dobré, aby tyto otázky obsahovaly už získané informace (pokud nějaké máme), hlavně proto, aby uživatel už ze začátku věděl, zda robot jím poskytnuté informace zpracoval správně. Toho můžeme dosáhnout tak, že soubor otázek bude vlastně souborem šablon pro otázky a do těchto šablon se při vytváření odpovědi na příslušná místa doplní již získané údaje.

Také tyto otázky je vhodné mít uložené v externím souboru a pro lepší přehlednost raději ve více. Soubor s otázkami pro rámec stavu *spoj* by mohl mít např. tento obsah:

```
to_place    >> kam chceš jet?
to_place    >> a kam bys chtěl jet?
to_place    >> a kam si přeješ jet?
from_place  >> odkud chceš jet <day> do <to_place:k1c2>?
day         >> kdy chceš jet z <from_place:k1c2> do <to_place:k1c2>?
hour        >> v kolik si přeješ odjet z <from_place:k1c2>?
hour        >> v kolik hodin chceš odjet z <from_place:k1c2>?
```

Řetězce vlevo jsou názvy slotů a v části za znaky „>>“ jsou šablony pro doplňující otázku na tyto sloty.

Výrazy v ostrých závorkách označují místa pro doplnění hodnoty z příslušného slotu. Tuto hodnotu je občas potřeba převést do jiné gramatické formy a je nutné, kromě názvu slotu, kterého hodnota se doplňuje, uvést také informaci o této formě.

Například chceme docílit toho, aby, pokud schází informace o místě odjezdu (které je reprezentováno slotem `to_place`) se nalezla šablona otázky pro tento slot a ta se doplnila o potřebná data (tedy den a cílovou stanici) v požadovaném tvaru. Šablona pro otázku na místo odjezdu

```
odkud chceš jet <day> do <to_place:k1c2>?
```

se tedy doplní například na

```
odkud chceš jet zítra do Hradce Králové?
```

2.7.3 Výsledek dotazu

Třetí skupinou odpovědí jsou odpovědi uživateli obsahující výsledek jeho požadavku. Odpovědi odesílené Taxatorem jsou většinou jen strohé strukturované informace. Nabízí se otázka, zda taxatorovu odpověď uživateli přeposlat tak, jak přišla, trochu upravit její formu, ale v zásadě informace poslat také strukturovaně, nebo se je pokusit uživateli předat v přirozeném jazyce.

Vysvětlím to na příkladě. Taxatorova odpověď na dotaz o vyhledání spoje z Brna do Prahy od osmi hodin vypadá takto:

```
Vypisuji 5 spojeni pro 25.4.2009 od 8:00:
*** 1.spoj:
(8:24) Brno hl.n. ( R 674 )
(12:04) Praha hl.n.
*** 2.spoj:
(8:30) Brno,,Benešova tř.hotel GRAND ( 182101 4)
(11:00) Praha,,ÚAN Florenc
*** 3.spoj:
(8:30) Brno,,Benešova tř.hotel GRAND ( 721309 27)
(11:00) Praha,,ÚAN Florenc
*** 4.spoj:
(8:41) Brno hl.n. ( EC 76 Gustav Klimt )
(11:29) Praha-Holešovice
*** 5.spoj:
(8:44) Brno hl.n. ( R 868 Slavkov )
(12:22) Praha hl.n.
```

Tato odpověď sice dokáže uspokojit uživatele, co se týče poskytnutí požadované informace, ale není to úplně ono. Lepší by bylo z Taxatorovy zprávy vybrat jen to podstatné, co uživatel chce vědět. Například odeslat uživateli jen první spoj:

```
Spojení z Brna do Prahy pro 25.4.2009 od 8:00:
(8:24) Brno hl.n. ( R 674 )
(12:04) Praha hl.n.
```

Další možností je si se zpracováním této zprávy pohrát ještě více a odeslat odpověď uživateli kupříkladu v takovémto formátu:

```
Nejbližším spojem z Brna do Prahy 25.4.2009 od 8:00 je vlak odjíždějící
v 8:24 ze stanice Brno hl.n.. V cílové stanici Praha hl.n. má být ve
12:04. Přeji příjemnou cestu!
```

Z těchto možných variant odpovědí se mi v tomto případě jeví jako nejlepší ta poslední, protože odpověď je napsána v přirozeném jazyce a robot tak působí více přátelsky.

Vytvoření takovéto odpovědi docílíme opět pomocí šablon. Tyto šablony budou mít podobnou strukturu jako šablony doplňujících otázek (2.7.2), jen se do nich budou místo hodnot slotů doplňovat data získaná z Taxatorovy zprávy.

Kapitola 3

Implementace

V této kapitole se nachází popis toho, jak je robot implementován. Nejdříve je zde popsáno jádro aplikace, které vše řídí, a v dalších částech jsou podrobněji popsány jednotlivé moduly a jejich funkce.

3.1 Jádro aplikace

Základním modulem programu je modul `main.py`. Po jeho spuštění se načtou pravidla přechodů (bude podrobněji popsáno v 3.2) a robot se připojí na server. Poté čeká na příchozí zprávy od uživatele. Jakmile dorazí zpráva, zjistí se, zda je pro odesilatele této zprávy vytvořen objekt třídy `Conversation`. Pokud takovýto objekt ještě není vytvořen, vytvoří se. Poté se zavolá metoda `handleMessage` objektu odesilatelovy konverzace, kterou si nyní popíšeme.

Tato metoda přijímá uživatelovu zprávu a jejím úkolem je na základě této zprávy vytvořit vhodný rámec (pokud ještě není vytvořen), naplnit sloty tohoto rámce informacemi získanými ze zprávy a vhodně uživateli odpovědět.

Příchozí zpráva je postupně srovnávána s vhodnými regulárními výrazy z pravidel z `transitions` — objektu obsahujícího všechna pravidla přechodů mezi stavy.

Pořadí, ve kterém se pravidla zkoušejí, je velmi důležité, protože se použije první pravidlo, kterému zpráva vyhovuje. Zkoušejí se pouze pravidla platná pro stav, ve kterém se robot právě nachází a pro stav *hello*, který tím, že je výchozím stavem, plní funkci jakéhosi „konverzačního“ stavu.

Nejdříve se tedy procházejí pravidla platná pro stav *hello* a zároveň vedoucí do tohoto stavu. To jsou ta „konverzační“. Dále se zkoušejí pravidla rovněž platná pro *hello*, ale vedoucí do stávajícího stavu. Po těch konečně přijdou na řadu pravidla platná pro stávající stav a nakonec se vyzkoušejí pravidla pro *hello*, ale vedoucí už do libovolného stavu.

Jakmile se narazí na pravidlo, kterému příchozí zpráva vyhovuje (vyhoví regulárnímu výrazu v pravidle) a úspěšně projde kontrolou morfologickým analyzátorem (popsanou v 3.4), tak se toto pravidlo použije. Pokud toto pravidlo vede do jiného stavu, než je ten současný, vytvoří se nový rámec odpovídající tomuto stavu. Ale ať už se vytváří nový rámec, či zůstal ten původní, pokusíme se naplnit sloty tohoto rámce. Hodnoty, kterými se naplňují sloty, dostaneme díky možnosti pojmenování zapamatovávaných skupin, kterou regulární výrazy v jazyce Python nabízejí. Tyto hodnoty se pak doplní do slotů rámce (struktura a chování rámců jsou podrobněji popsány v části 3.3).

Na základě toho, jaký rámec je aktivní a které jeho sloty ještě nejsou doplněny, se buď

vytvoří otázka pro uživatele, dotazující se na hodnotu slotu, který je potřeba doplnit, nebo pokud již jsou všechny potřebné sloty doplněny, vytvoří se dotaz pro Taxatora, počká se na jeho odpověď, a ta se odešle uživateli.

3.2 Načtení gramatiky a pravidel přechodů

O načtení gramatiky a pravidel přechodů se stará modul `transitions.py`. Obsahuje jednu hlavní třídu `Transitions`, která je inicializovaná jako typ slovník sloužící pro uložení pravidel přechodů mezi stavy. Tato třída také obsahuje slovník s gramatickými pravidly `patterns` a metody pro nahrávání pravidel.

3.2.1 Načtení gramatických pravidel

Při inicializaci této třídy se nejprve načtou gramatická pravidla (viz příloha B.1). Ta jsou uložena v jednom externím souboru a postupně se z něj čtou. Pokud se na pravé straně pravidla vyskytují nějaké neterminály, tak se tyto neterminály podle předem nahraných pravidel přepíší a takto nahrazené pravidlo se přidá do `patterns`.

Například po načtení těchto tří pravidel:

```
<town_name>   => ([-\w]+ ?)+
<to_place>    => <town_name>
<to_place_p>  => do <to_place>
```

bude v `patterns` neterminálům `town_name` a `to_place` přiřazen výraz „`([-\w]+ ?)+`“ a výraz pro `to_place_p` bude „`do ([-\w]+ ?)+`“.

Proces načítání ale komplikuje několik vlastností těchto pravidel.

Jednou z nich je přítomnost značek pro morfologickou analýzu. Tyto značky jsou připojeny k některým neterminálům a je potřebné někde zaznamenat, jaká značka ke kterému úseku výsledného výrazu patří. Bylo třeba tedy zajistit dvě věci — vyznačit ve výrazu tu část, ke které značka patří a také si tuto značku pro daný úsek a pravidlo zapamatovat.

Možnost vyznačení určitých částí ve výrazu je nutná také kvůli zapamatování si hodnot potřebných pro dotaz na službu (v gramatických pravidlech jsou zapsány jako neterminály začínající znakem „`!`“). K označení těchto částí jsem využila možnosti regulárních výrazů a vložila je do „pojmenované“ skupiny „`?P<`*`název`*`>`“.

Tohoto pojmenování určitých částí využívám právě také kvůli vyznačení úseku, který je potřeba zkontrolovat morfologickým analyzátorem (úseku vyjádřenému neterminálem se značkou). Pokud je tento neterminál označen znakem „`!`“ kvůli zapamatování si hodnoty, je situace jednoduchá a skupinu si v regulárním výrazu pojmenuji podle názvu neterminálu. Problém ale nastává, pokud neterminál takto označen není. Označování skupiny podle názvu neterminálu by značně omezilo možnosti použití některých neterminálů, protože v regulárním výrazu musí být všechny názvy skupin jedinečné, takže by tento neterminál musel být v jednom pravidle také jedinečný. Což by znemožnilo například častější použití `<s:`*`značka`*`>` jako neterminálu označujícího libovolné slovo (které se skrývá pod `<s>`) vyhovující gramatickým kategoriím určeným zadanou značkou. Tento problém řeším tak, že tyto skupiny nenazývám podle názvu neterminálu, ale zavádím pro ně v rámci výrazu jednoznačné názvy — postupně je pojmenovávám jako `L1`, `L2`, `L3`... Občas se stane, že více výrazů s takto pojmenovanými skupinami je později vloženo do jiného výrazu. V těchto případech možné kolize řeším tak, že tyto skupiny přejmenuji tak, aby zase všechny tvořily jednu posloupnost `L1`, `L2`, `L3`... Například když jednomu neterminálu je přiřazen výraz

obsahující skupiny `L1` a `L2` a druhému výraz se skupinou `L1`, skupiny se přepíší tak, aby výsledný výraz obsahoval skupiny `L1`, `L2` a `L3`.

Dvojice název skupiny – značka pak pro daný výraz ukládám do slovníku `tags` a výsledný výraz spolu s tímto slovníkem vkládám pod příslušným klíčem do `patterns`.

Další vlastností, kterou bylo třeba řešit, bylo to, že pro jeden neterminál může být definovaných více přepisovacích pravidel. Vyřešila jsem to tak, že v `patterns` se pod každým klíčem (kterým je název neterminálu) nachází ne pouze dvojice výraz – značky, ale je tam seznam těchto dvojic. Později při přepisování tímto neterminálem se musí pamatovat na to, že možností přepsání může být více. Pokud je, množství výsledných výrazů se znásobí tak, aby existovaly všechny možnosti těchto výrazů.

A ačkoliv se to nezdá, komplikací byly také nepovinné neterminály — tedy ty, které zapisujeme se znakem „`?`“ na začátku. Přepsat tuto skupinu na nepovinnou je v regulárních výrazech jednoduché, jen se za ni napíše znak „`?`“, ale potíž byla s okolními mezerami. Pokud okolní mezery nebyly zahrnuty do nepovinné skupiny, tak při absenci nepovinného neterminálu byly vyžadovány dvě mezery po sobě. A to, že si nepovinný neterminál přivlastní jednu sousední mezeru, také nic nevyřešilo. Nakonec se mi osvědčilo všechny možné výskyty mezer nahradit výrazem „`\b[ ]*`“, který povoluje libovolnou délku řetězce mezer (tedy i nulovou), ale zároveň se musí vyskytovat na hranici slova.

3.2.2 Načtení pravidel přechodů

Jakmile se načte gramatika do `patterns`, provede se podobná věc s pravidly přechodů. Ty jsou pro lepší přehlednost uloženy ve více souborech, podle toho, do kterého stavu se přechází — stav, do kterého se po použití pravidla přejde, je určen názvem tohoto souboru.

Pravidla přechodů se ukládají do třídy `Transitions`, ke které v tomto případě přistupujeme jako k datovému typu slovník. Načítání těchto pravidel probíhá tak, že se postupně čtou soubory s pravidly. V části pravidla, která je zapsaná pomocí naší gramatiky, se nahradí neterminály způsobem popsaným v předchozí podkapitole a celé pravidlo se přidá do `Transitions`. Pravidla jsou uložena tak, že výchozí stav přechodu je klíčem a hodnotou pod tímto klíčem je seznam obsahující regulární výraz, stav, do kterého se po použití pravidla přejde a nepovinně odpověď uživateli (viz 2.7.1).

3.3 Rámce, sloty, odpovědi

V této podkapitole si podrobněji popíšeme strukturu rámce a jeho funkce.

Rámce jsou definovány v modulu `frames.py`. Pro každý druh rámce je zde vytvořena zvláštní třída, ale všechny tyto třídy dědí z třídy `Frame`.

Velmi důležitým atributem rámce jsou sloty. Kvůli lepšímu provázání s daty v textových souborech jsou tyto sloty uloženy ve slovníku, ale jejich počet i názvy jsou pevně dány pro každý rámec.

Při inicializaci třídy rámce se pro tento rámec vytvoří sloty (objekty třídy `Slot`) a do nich se z externích souborů nahrají doplňující otázky pro uživatele (viz 2.7.2).

Každý slot uchovává následující informace:

- hodnotu
- seznam šablon otázek dotazujících se na hodnotu slotu
- stav slotu (výchozí, dotazující se, doplněný)

Stav slotu udává informaci o tom, zda již byla do slotu doplněna hodnota, nebo jestli se na tuto hodnotu zrovna dotazujeme. Stav *dotazující se* nemá velkou váhu, slouží pouze pro upřesnění v některých případech, kdy se podle formátu uživatelovy zprávy nedá rozpoznat, který slot se má danou hodnotou doplnit. Uživatel díky tomu není nucen podávat pouze ty informace, na které je zrovna dotazován.

Každý rámec vlastní metody `check`, `update` a `answer`.

Metoda `check` provádí dodatečné kontroly vstupních dat, které nebylo možné provést ještě před použitím pravidla, protože jsou specifické pro každý rámec — například kontrola toho, jestli vstup, který má představovat jazyk, do kterého se má překládat, je skutečně nějaký známý jazyk, nebo kontrola, jestli slovo vyjádřující hodinu lze převést na číselnou hodnotu.

Metoda `update` ukládá vstupní data do jednotlivých slotů. Některé hodnoty musí nejdříve převést do požadovaného formátu, například vstupní hodnotu „angličtiny“ uloží do slotu pouze jako zkratku „en“, nebo hodinu zapsanou slovně jako „deváté“ uloží jako „9:00“ Také musí umět převádět názvy měst z různých pádů (hlavně z genitivu a lokálu) do nominativu. Ke všem výše vyjmenovaným úkolům využívá funkcí morfologického analyzátoru popsaných v podkapitole 3.4.

Metoda `answer` se stará o vytvoření odpovědi pro uživatele. Pokud při jejím zavolání jsou všechny potřebné sloty doplněné, pošle požadavek Taxatorovi. Pokud hodnota v nějakém slotu chybí, zavolá pro tento slot metodu `makeQuestion`, jejímž úkolem je vytvoření otázky dotazující se uživatele na hodnotu tohoto slotu.

Metoda `makeQuestion`, kterou všechny rámce dědí z třídy `Frame`, pro zadaný slot vybere šablonu otázky a na určených místech ji doplní již známými hodnotami slotů daného rámce. Pokud pole označující místa pro doplnění obsahují značku pro morfologický analyzátor, hodnotu doplňovanou na toto místo je nutné převést do tvaru určeného značkou.

Ukážeme si, jak vypadá rámec po uživatelově zprávě „Chtěla bych jet zítra v deset do Hradce Králové“. Stav, do kterého se po této zprávě přejde, se zjistí podle pravidla, kterému zpráva vyhoví, v tomto případě je to stav *spoj* a vytvoří se rámec pro vyhledávání spojení s příslušnými sloty. Tyto sloty budou po doplnění poskytnutých hodnot metodou `update` vypadat takto:

`from_place`:

- hodnota:
- seznam otázek:
  - `odkud chceš jet <day> do <to_place:k1c2>?`
  - `z jakého města chceš jet <day> do <to_place:k1c2>?`
- stav: dotazující se

`to_place`:

- hodnota: `Hradec Králové`
- seznam otázek:
  - `kam chceš jet?`
  - `a kam bys chtěl jet?`
  - `a kam si přeješ jet?`
- stav: doplněný

`day`:

- hodnota: `zítra`
- seznam otázek:
  - `ve který den chceš jet z <from_place:k1c2> do <to_place:k1c2>?`
- stav: doplněný

`hour`:

- hodnota: `10:00`
- seznam otázek:
  - `v kolik si přeješ odjet z <from_place:k1c2>?`
  - `v kolik hodin chceš odjet z <from_place:k1c2>?`
- stav: doplněný

Z ekologických důvodů jsou pro ukázku zvoleny jen čtyři sloty, ve skutečnosti je jich využáváno více.

Můžete si všimnout, že uživatelka sice napsala „v deset“, ale do slotu se doplnilo „10:00“, a také cílová stanice „Hradce Králové“ se uložila jako „Hradec Králové“.

Je vidět, že chybí už jen hodnota výchozí stanice. Je tedy třeba vytvořit otázku dotazující se na tuto hodnotu. O to se stará výše zmíněná metoda `makeQuestion`. Řekněme, že tentokrát bude náhodně vybrána tato šablona pro otázku:

```
z jakého města chceš jet <day> do <to_place:k1c2>?
```

a tato se pomocí funkcí využívajících morfologický analyzátor, které převedou název města do požadovaného tvaru, doplní na:

```
z jakého města chceš jet zítra do Hradce Králové?
```

Teď už stačí jen odeslat tuto otázku uživateli a počkat na jeho odpověď.

3.4 Funkce morfologické analýzy

Funkce zabývající se morfologickou analýzou se nacházejí v modulu `ma.py`. Všechny tyto funkce využívají modul *pylibma*, který je rozhraním pro jazyk Python pro knihovnu *libma* [11]. Z modulu pylibma používám tyto funkce:

`Morph` – pro zadané slovo vrací seznam trojic obsahujících základní tvar slova, anotaci a vzor.

`AllWords` – pro zadané slovo v základním tvaru vrátí seznam obsahující všechny možné formy tohoto slova

`GetValue` – vrátí číselnou hodnotu posledně analyzovaného slova

Funkce nacházející se v mnou vytvořeném modulu `ma.py`, které stojí za zmínku:

`toLemma` – tato funkce na základě vstupu, kterým je slovo a značka s jeho gramatickými kategoriemi, vrátí základní tvar (lemma) tohoto slova. Využívá funkci `Morph` z pylibma. Používá se pro převedení slova, které je v nějaké formě, do základního tvaru. Například pro slovo „angličtiny“ vrátí „angličtina“ a až na základě této formy se rozpoznává, o jaký jazyk se jedná.

`getAnnot` – vrátí anotaci (značku s gramatickými kategoriemi) slova word. Tato funkce rovněž používá funkci `Morph`.

`getForm` – převede slovo zadané v základním tvaru do tvaru určeného značkou. Tato funkce měla využívat funkci `GetForm` z pylibma, ale tato mi pro některá slova nefungovala, využívám proto místo ní funkci `GetForm2`.

`GetForm2` – náhrada za `GetForm` z pylibma, vrací seznam možných tvarů slova zadaného v základním tvaru vyhovujícím zadané značce. Používá funkci `AllWords` z pylibma.

`getPhraseForm` – převede název města z určitého pádu do jiného. Používá se nejčastěji k převodu názvů měst z nominativu do jiného pádu nebo opačně.

3.5 Komunikace se službami

Jelikož si náš robot nezískává informace o službách sám přímo z databáze nebo analýzou webových stránek, ale ptá se na ně jiného robota, vyskytl se problém v tom, že potřebujeme nějak rozhodnout, kterému uživateli je určena příchozí odpověď. Vyřešila jsem to tak, že když je v nějaké konverzaci odeslán požadavek pro Taxatora, tak objekt této konverzace vložím do fronty a pokud přijde odpověď, předpokládám, že patří první konverzaci ve frontě. Někdy se ale může stát, že odpověď od Taxatora nepřijde, a tak je doba pobytu konverzace ve frontě omezena časovým limitem.

Dále jsem v návrhu popsala, v jakém formátu by se informace získané od Taxatora měly poslat uživateli. Toto ale dosud nebylo implementováno, protože vývoj aplikace se soustředil hlavně na to, aby robot dokázal uživatele správně pochopit a tak pro něj získat požadované informace. To, zda jsou informace odeslány ve více nebo méně líbivém formátu, není v porovnání se schopností porozumět uživateli tak důležité a je možné tuto funkci kdykoliv implementovat.

Kapitola 4

Testování

V této kapitole je popsáno, jak byl výsledný program testován a jaké prostředky byly použity pro ukládání a vyhodnocování rozhovorů. Jsou zde uvedeny také výsledky testování na uživatelích.

4.1 Nasazení v testovacím provozu

Vytvořený systém byl spuštěn v testovacím provozu a předveden určitému množství lidí. Cílem tohoto testování je zjistit, jak si lidé vedou v komunikaci s robotem, zda robot rozumí jejich požadavkům, jestli se různí lidé ptají stejně, nebo každý jinak a v neposlední řadě je cílem i to, aby se přišlo na chyby v implementaci. Testování více lidmi je v tomto případě velmi důležité, protože v přirozeném jazyce lze tentýž dotaz vyjádřit mnoha způsoby a čím více lidí testuje, tím více možností vyjádření může být vyzkoušeno. Toto testování pomáhá doladit nedostatky hlavně v návrhu gramatiky.

Pro testování byl vytvořen účet `freddie@jabbim.cz` na serveru Jabbim. Pod tímto účtem je robot spuštěn na jednom „spřáteleném serveru“. Mohl by být spuštěn kdekoliv, ale pro robota je vhodné, aby byl dostupný prakticky nepřetržitě, aby mohl být uživatelům k dispozici kdykoliv ho potřebují nebo si na něj vzpomenou.

4.2 Záznam konverzací

Všechny konverzace uživatelů s robotem i robota s robotem poskytujícím služby jsou zaznamenávány.

Z důvodu lepšího vyhodnocování a hledání chyb je potřebné, aby v uložených konverzacích bylo možné jednoduše vyhledávát. Jako způsob uložení dat byla proto zvolena databáze, konkrétně sqlite3, protože je hodně rozšířená a hlavně snadno přenosná. Je tak možné spolu s přenesením robota na jiný server jednoduše přenést i historii jeho konverzací s uživateli.

4.2.1 Schéma databáze

Schéma této databáze je jednoduché. Databáze obsahuje dvě tabulky: `conversation` a `message`. Tabulka `conversation` obsahuje sloupce `id` (klíč pro identifikaci), `jid` (Jabber ID uživatele, který s robotem vede konverzaci) a `bot_version` (verze robota) a slouží k ukládání informací o celé konverzaci. Tabulka `message` má sloupce `id`, `datetime` (informace o čase),

`sender` (odesilatel - „user“, „bot“, nebo „taxator“), `content` (vlastní obsah zprávy), `note` (poznámka) a `conversation_id` (odkaz na konverzaci, ke které zpráva patří) a slouží k ukládání obsahu zpráv patřících k dané konverzaci.

4.2.2 Prohlížeč uložených konverzací

Jak už jsem se zmínila výše, konverzace jsou zaznamenávány v databázi. S touto formou reprezentace dat se sice dobře manipuluje, ale pro prohlížení a zkoumání jednotlivých konverzací sama o sobě není příliš vhodná. Z tohoto důvodu jsem si vytvořila jednoduchou webovou stránku sloužící k prohlížení uložených konverzací.

4.3 Vyhodnocení reakce uživatelů

Výsledný systém byl testován na vzorku asi 20 uživatelů. Při vyhodnocování testovacích konverzací jsem se zaměřila na tyto oblasti:

Začátek komunikace

Jak si uživatelé vedli při prvních pokusech o komunikaci s robotem? Kolik z nich vědělo, jak se zeptat?

Naprostá většina uživatelů už ze začátku věděla, jak s robotem komunikovat – psát běžné věty. Pouze 3 uživatelé zadávali jen hesla, a ta robot většinou nechápal. Někteří jiní uživatelé se k heslům uchylovali později, když robot nepochopil jejich požadavek napsaný větou. Tyto výsledky ale nejsou příliš vypovídající, protože různí uživatelé měli různé počáteční informace o robotovi a to samozřejmě mělo vliv na jejich způsob vyjádření požadavku.

Forma odpovědí na otázky robota

Jak uživatelé odpovídali na otázky robota? Skloňovali názvy měst nebo je zadávali v prvním pádu?

Většina uživatelů (63%) odpovídala na doplňující otázky částmi vět a názvy měst skloňovala. Například na otázku „kam chceš jet?“ odpovídali „do Brna“ (nebo jiného města). Zbytek odpovídal hesly — jen „Brno“. Byli ale i tací, kteří nejdříve odpovídali hesly a až zjistili, že robot tyto hesla chápe, rozepsali se trochu více a začali odpovídat částmi vět.

Reakce na nepochopení

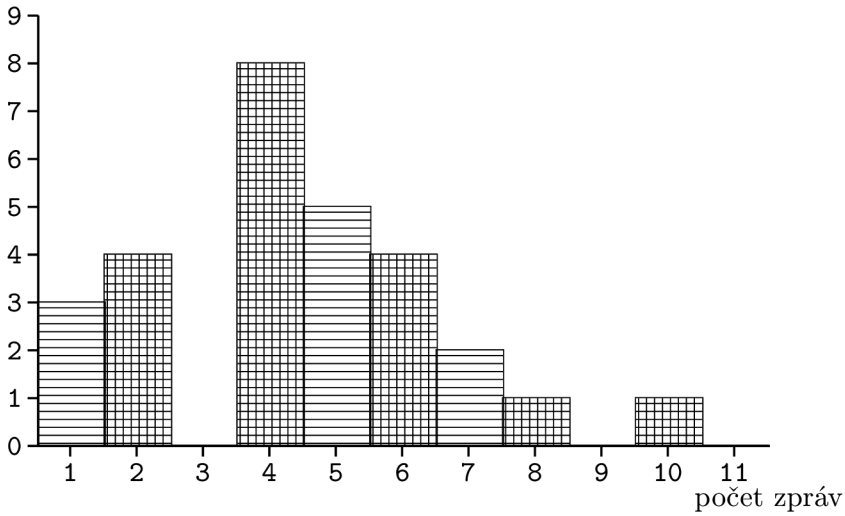
Jak uživatelé reagovali, když je robot nepochopil správně?

Pokud robot nerozuměl uživatelově zprávě, byli uživatelé většinou hodně trpěliví a zkoušeli svoje požadavky napsat jinak. Jiná situace byla, když robot například špatně pochopil název města, nebo slovo, které chtěl uživatel přeložit. Uživatelé sice viděli, že je robot špatně pochopil (protože špatně pochopené informace použil v doplňující otázce), ale většina toto ignorovala a odpovídala na doplňující otázky, které jim robot kladl, a až poté, co jim robot vrátil nežádoucí výsledek, to začali řešit. Většinou tak, že se pokoušeli robota přesvědčit, že chtěli něco jiného (někdy úspěšně, někdy bohužel ne). Přibližně čtvrtina uživatelů začala robotovi nadávat.

Počet zpráv potřebných k získání požadované informace

Po kolika odeslaných zprávách uživatel získal žádanou informaci?

Počet zpráv, které musel uživatel poslat, aby získal požadovanou informaci, byl různý. Zatímco k použití slovníku stačila převážně jedna, nebo dvě zprávy, u vyhledávání dopravních spojení byla situace jiná, protože pro vyhledání spoje je potřeba získat mnohem více vstupních informací. Na obrázku 4.1 je histogram znázorňující četnost toho, po kolika zprávách uživatelé získali žádanou informaci. Jsou zde započítány jen úspěšné pokusy o vyhledání spojení. Také zde nepočítám hledání spojení, pro které byly použity některé informace z předešlého hledání a uživatel pouze upravil svůj požadavek.



Obrázek 4.1: Vyhledávání spojení. Histogram znázorňující počet zpráv uživatele, po kterých obdržel požadovanou informaci.

Neúspěšných pokusů bylo bohužel více, než těch úspěšných (57%). Pozitivní ale je, že až 80% neúspěšných pokusů bylo zapříčiněno jedním ze tří hlavních nedostatků. Rozdělení neúspěšných pokusů podle příčiny, kvůli které byly neúspěšné, je ukázáno na grafu 4.2.

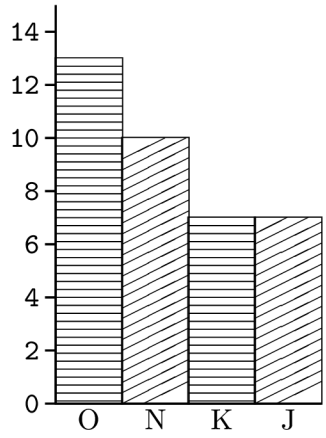
Popis jednotlivých sloupců grafu:

O – neúspěšné pokusy o vyhledání spoje zapříčiněné tím, že některé názvy měst nebo obcí se v České republice vyskytují vícekrát a Taxator žádal o upřesnění stanice. Toto upřesňování stanice ale v době testování ještě nebylo v robotu implementováno.

N – příčinou neúspěchu bylo dotazování se na neznámou stanici, většinou způsobené špatným pochopením uživatelova požadavku. Bylo by možné to řešit důslednější kontrolou morfologickým analyzátorem a stanice, které ve slovníku morfologického analyzátoru nejsou, by se dalo ošetřit tak, že robot vyzve uživatele, aby zadal jen název stanice.

K – neúspěch byl zapříčiněn nepřesně formulovanou doplňující otázkou „kdy chceš jet... “ dotazující se na den odjezdu. Uživatelé často odpovídali zadáním času odjezdu, ale robot se tvářil, že je nechápe a pořád se ptal na „kdy... “. Tuto chybu jsem jednoduše opravila přepsáním otázky na „ve který den chceš jet... “.

J – jiné příčiny neúspěchu



Obrázek 4.2: Vyhledávání spojení. Neúspěšné pokusy.

Problémy

Co činilo největší problémy? S čím si robot nevěděl rady?

Testování, kromě nedostatků popsaných výše, odhalilo také problém s přepínáním služeb. To je způsobeno tím, že pravidla, po kterých se změní využívaná služba, se zkoušejí až jako poslední a uživatelova zpráva, která vyhovuje některému z těchto pravidel, vyhoví jinému pravidlu, více obecnému. Řešením by mohlo být tato pravidla zpřísnit natolik, aby mohla být zkoušena dříve.

Kapitola 5

Závěr

Výsledkem mé práce je robot komunikující v přirozeném jazyce, který slouží jako rozhraní pro službu vyhledávání vlakových a autobusových spojení a slovník. Tyto služby implementuje robot Taxator serveru Jabbim.

Za zajímavé řešení považuji generování pravidel, která zajišťují porozumění uživatelově zprávě, pomocí kombinace bezkontextové gramatiky a regulárních výrazů. Díky této kombinaci robot rozumí i s poměrně malým množstvím gramatických pravidel. Použití bezkontextové gramatiky umožňuje libovolně kombinovat pořadí různých částí vět, za to regulárními výrazy se lépe zapisují různé varianty koncových slov.

Jsem ráda, že jsem díky této práci alespoň trochu nahlédla do oblasti zpracování přirozeného jazyka. Tato oblast mi přijde zajímavá a chtěla bych se jí dále věnovat.

5.1 Další vývoj

Práce na tomto projektu mě bavila a určitě plánuji svého robota dále vylepšovat a přidávat nové funkce, aby byla radost ho používat. Věcí, které by se dalo vylepšit, je mnoho.

Jednou z nich je vyřešit potíže se změnou využívané služby, protože v současné implementaci se stává, že bývá problém přejít ze stavu překládání do stavu hledání spojení. Lepšího výsledku by snad šlo dosáhnout zdokonalením pravidel přechodů mezi stavy a úpravou pořadí, v jakém se pravidla zkoušejí.

Další věcí, kterou by bylo dobré implementovat, je to, aby robot rozuměl i bez diakritiky, protože velké množství českých uživatelů je zvyklé psát bez ní. Podpora této formy vyjádření požadavků by proto slušnému robotovi neměla chybět. Implememtace však může být komplikovaná, protože morfologický analyzátor přijímá pouze slova se správně zapsanou diakritikou. Jako řešení mě napadá buď vytvořit alternativní slovník pro morfologický analyzátor pro slova bez diakritiky, nebo se pokusit potřebnému slovu diakritiku doplnit (tuto funkci poskytuje např. morfologický analyzátor Ajka). Také je zde možnost nastudovat implementaci používaného morfologického analyzátoru a pokusit se ho upravit tak, aby slova s chybějící diakritikou přijímal.

Samozřejmostí je také doplnit další služby, pro které by robot sloužil jako rozhraní. Mohl by umět například podávat informace o televizním programu, o počasí nebo hledat informace na serverech Google nebo Wikipedii.

Literatura

[1] Dialog system. Wikipedia [online], [cit. 2009-04-30]. URL http://en.wikipedia.org/wiki/Dialog_system

[2] ELIZA. Wikipedia [online], [cit. 2009-04-29]. URL http://en.wikipedia.org/wiki/ELIZA

[3] Extensible Messaging and Presence Protocol. Wikipedia [online], [cit. 2009-04-29]. URL http://en.wikipedia.org/wiki/XMPP

[4] Konečný automat. Wikipedie [online], [cit. 2009-05-23]. URL http://cs.wikipedia.org/wiki/Konečný_automat

[5] Regulární výraz. Wikipedie [online], [cit. 2009-05-23]. URL http://cs.wikipedia.org/wiki/Regulární_výraz

[6] Josef Psutka et al.: *Mluvíme s počítačem česky*. Academia, 2006, ISBN 80-200-1309-1, 752 s.

[7] Alexander Meduna and Roman Lukáš: *Formální jazyky a překladače*. Studijní opora, FIT VUT v Brně, 2006.

[8] Radek Sedláček: *Morfologický analyzátor češtiny*. Diplomová práce, FI MU v Brně, 1999.

[9] Radek Sedláček: Ajka tagset. [online], březen 2006, [cit. 2009-05-22]. URL http://nlp.fi.muni.cz/projekty/ajka/tags.pdf

[10] František Zbořil and František Zbořil: *Základy umělé inteligence*. Studijní opora, FIT VUT v Brně, 2006.

[11] Stanislav Černý: *Morfologický analyzátor pomocí konečných automatů*. Bakalářská práce, FIT VUT v Brně, 2008.

Seznam příloh

Dodatek A

Ukázky dialogů

A.1 Vyhledání spoje

```
uživatel:  Ahoj!
robot:     ahoj, co bys rád?
uživatel:  Zjisti mi, prosím, kdy nejříve jede autobus do Brna!
robot:     odkud chceš jet do Brna?
uživatel:  Z Prahy.
robot:     kdy chceš jet z Prahy do Brna?
uživatel:  Dnes.
robot:     v kolik si přeješ odjet z Prahy?
uživatel:  Co nejdříve.
robot:     Vypisuji 4 spojeni pro 25.4.2009 od 20:23:
           *** 1.spoj:
           (20:25) Praha,,ÚAN Florenc ( 000212 1 )
           (23:00) Brno,,ÚAN Zvonařka
           *** 2.spoj:
           (20:30) Praha,,ÚAN Florenc ( 000232 6)
           (22:55) Brno,,ÚAN Zvonařka
           *** 3.spoj:
           (20:30) Praha,,ÚAN Florenc ( 000008 3)
           (23:10) Brno,,ÚAN Zvonařka
           *** 4.spoj:
           (21:00) Praha,,ÚAN Florenc ( 000539 9)
           (23:30) Brno,,Benešova tř.hotel GRAND
           *** 5.spoj:
           (22:35) Praha,,ÚAN Florenc ( 000015 2 )
           (1:20) Brno,,ÚAN Zvonařka
uživatel:  Dík!
robot:     není zač
```

Dodatek B

Ukázky pravidel

B.1 Gramatická pravidla

```
<slovo>          => \w+
<s>              => \w+
<predlozka>      => kolem|v|ve|okolo|k|před|po|od|o
<komu>           => mi|mě|mně|nám|jim
<dik>            => díky|dík|děkuji|thx
<pozdrav>        => ahoj|zdar|nazdar|čau
-----SPOJ-------------------------------------------------------------
-----casove udaje-----------------------------------------------------
<den>            => dneska|zítra|dnes|(\d{1,2}\.\d{1,2}\.(\d{2,4})?)
<hodina>         => (\d{1,2}([:.]\d{1,2})?|(půl)? <s:k4>)
<hodina_p>       => <?predlozka> <!hodina> (h\.?|hodinu?)?
<dennidoba>      => ráno|dopoledne|poledne|odpoledne|večer
<dennidoba_p>    => <?predlozka> <!dennidoba>
<cas>            => <hodina_p> <dennidoba_p>
<cas>            => <dennidoba_p> <hodina_p>
<cas>            => <dennidoba_p>|<hodina_p>
<ted>            => nejbližší|příští|teď|co nejdříve?
<kdy>            => <!den> <cas>
<kdy>            => <cas> <!den>
<kdy>            => <!ted>|<!den>|<cas>
-----------------------------------------------------------------------
<nazev_mesta>    => ([-\w]+ ?)+
<odkud>          => <nazev_mesta>
<kam>            => <nazev_mesta>
<kam_p>          => do <!kam:k1c2>
<odkud_kam>      => (ze? <!odkud:k1c2>)? <kam_p>
<den_odkud_kam>  => <!den> <odkud_kam>
<kdy_odkud_kam>  => <odkud_kam> <kdy>
<kdy_odkud_kam>  => <kdy> <odkud_kam>
<kdy_odkud_kam>  => <odkud_kam>
<mesto>          => <nazev_mesta>
```

Dodatek C

Obsah přiloženého DVD

- `src` – adresář, obsahuje zdrojové soubory programu
  - `grammar` – adresář, obsahuje soubory s pravidly
  - `libma` – adresář, obsahuje knihovnu pro morfologický analyzátor češtiny
  - `logs` – adresář, obsahuje databázi pro záznam konverzací
  - `questions` – adresář, obsahuje soubory s doplňujícími otázkami
  - `dict.conf` – konfigurační soubor pro libma
  - `frames.py`
  - `logging.py`
  - `ma.py`
  - `main.py`
  - `transitions.py`
- `logviewer` – adresář, obsahuje php skript pro prohlížení uložených konverzací
- `zprava` – adresář, obsahuje zdrojové kódy této zprávy
- `README` – soubor s návodem pro instalaci a spuštění programu