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

Diplomová práce

Chatovací robot

Plzeň, 2012 Stanislav Strnad

Originál zadání práce (ten s červeným kulatým razítkem) v jednom výtisku práce, kopie zadání ve druhém.

Prohlášení

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

V Plzni dne _ _._ _._ _ _ _ Stanislav Strnad, _ _ _ _ _ _ _ _

Abstract

Chatting robot

This thesis conceives a chatting robot based on Artificial Intelligence Markup Language (AIML), an XML dialect for creating natural language software agents. This work can be divided as follows. In the first chapter, we introduce the topic of chatting robots and briefly outline their functionality. The second chapter contains a description of tools and methods for network applications design. The third chapter compares methods for generating uninformed dialogues. In the fourth chapter, the theory of the chatting robot and its key features are presented and explained. In the fifth chapter, the actual implementation of a chat robot, its structure and functionality are presented. The sixth chapter presents and evaluates results obtained from our experiments. Finally, the thesis is summarized in the seventh chapter where further extensions to our architecture are suggested.

Obsah

1 Úvod

Cílem této práce je seznámení se s nástroji a metodami pro tvorbu síťových aplikací a zhodnocení těchto nástrojů, dále seznámení se s metodami generování neinformovaně vedených dialogů a s jazykem AIML (Artificial Intelligence Markup Language). Dalším cílem je pak návrh a implementace aplikace, která na základě uživatelského vstupu dokáže automaticky vygenerovat odpovídající výstup. Touto aplikací je chatovací robot, který konverzuje s uživateli jim srozumitelným přirozeným jazykem.

Tato práce je rozdělena do následujících kapitol. První kapitola představuje téma chatovacích robotů a krátce naznačuje jejich funkcionalitu. V druhé kapitole se nachází popis nástrojů a metod pro tvorbu síťových aplikací. Třetí kapitola je věnována metodám pro generování neinformovaných dialogů. V další kapitole je teoreticky popsán chatovací robot a jeho vlastnosti. Pátá kapitola se zabývá implementovaným chatovacím robotem, jeho strukturou a funkcionalitou. V další kapitole jsou uvedeny a zhodnoceny výsledky testování vytvořené aplikace. V závěru jsou popsány možné návrhy na rozšíření aplikace a práce je celkově zhodnocena.

2 Nástroje a metody pro tvorbu síťových aplikací

V této práci popisovaná aplikace je založena na komunikaci *klient/server*. Klient/server komunikace je standardním modelem obsahujícím dva typy účastníků - *klienty*, kteří generují požadavky, a *server*, který tyto požadavky zpracovává, resp. generuje odpovědi. V této kapitole jsou popsány možnosti pro tvorbu síťových aplikací typu klient/server.

2.1 Java

Java je objektově orientovaný programovací jazyk vytvořený firmou Sun Microsystems v roce 1995. Díky přenositelnosti a dalším výhodám (viz.níže) je Java jedním z nejpoužívanějších programovacích jazyků.

2.1.1 Výhody programovacího jazyka Java

- Je navržen pro podporu aplikací v síti,
- přenositelnost – díky vytváření tzv. *bajtkódu (byte-code)*, kódu nezávislém na architektuře počítače, lze program spustit na jakémkoliv stroji, disponujícím běhovým prostředí *Java Virtual Machine (JVM)*,
- je určen pro psaní spolehlivého softwaru,
- správa paměti je realizována automaticky pomocí *Garbage Collectoru,* součástí JVM spravujícím paměť.
- bezpečnost (např. ve srovnání s jazykem Pascal).

2.1.2 Nevýhody programovacího jazyka Java

- Programy psané v Javě mohou být podstatně pomalejší než nativně běžící programy,
- programy napsané v Javě mohou mít větší pamětovou náročnost.

2.1.3 Programování síťových aplikací v Javě

Pro vytváření síťových aplikaci obsahuje Java dvě možnosti:

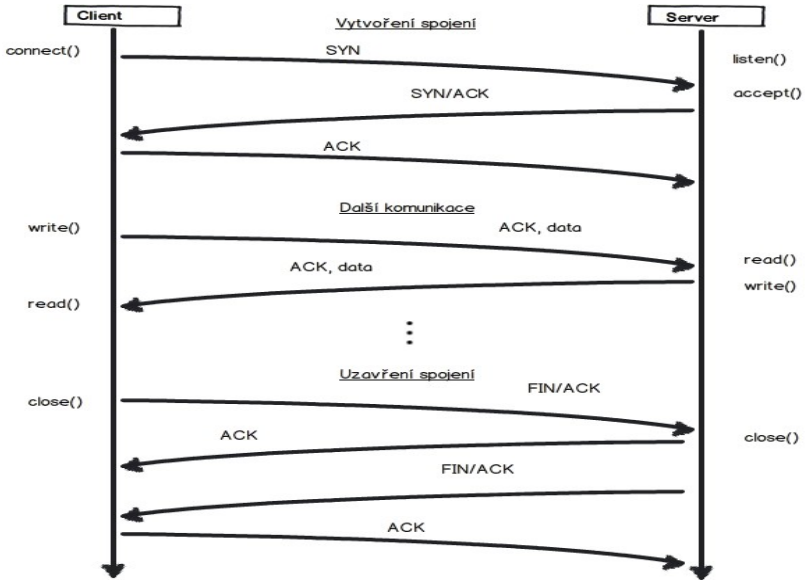
Komunikace přes sockety a protokoly TCP, UDP

Vysvětleme nejprve termín *sockety*. Sockety jsou koncovým bodem v obousměrném komunikačním kanálu zprostředkovávající, prostřednictvím síťových protokolů (např. TCP a UDP), komunikaci mezi procesy na stejném nebo na různých počítačích. Vývoj aplikací komunikujících přes sockety se nazývá *socketové programování*.

Protokoly TCP a UDP patří do rodiny protokolů *TCP/IP,* obsahující sadu protokolů pro komunikaci v počítačových sítích. Hlavním rozdílem mezi těmito protokoly je, že spojově orientovaný protokol *TCP* narozdíl od nespojově orientovaného protokolu *UDP* zaručuje spolehlivý přenos dat. Spojově orientovaný protokol potřebuje k navázání komunikace tzv. *handshaking* – výměna potvrzovacích zpráv. Spolehlivý přenos dat protokolem TCP je zajištěn potvrzováním o přijetí dat a možností opětovného zasílání dat. Nespojově orientovaný protokol UDP je založený na odesílání nezávislých zpráv a neumožnuje ověřit, zda data správně dorazila příjemci.

Socket je reprezentován třídou `java.net.socket`, která obsahuje mechanismus naslouchání klientům a správu spojení s klienty.

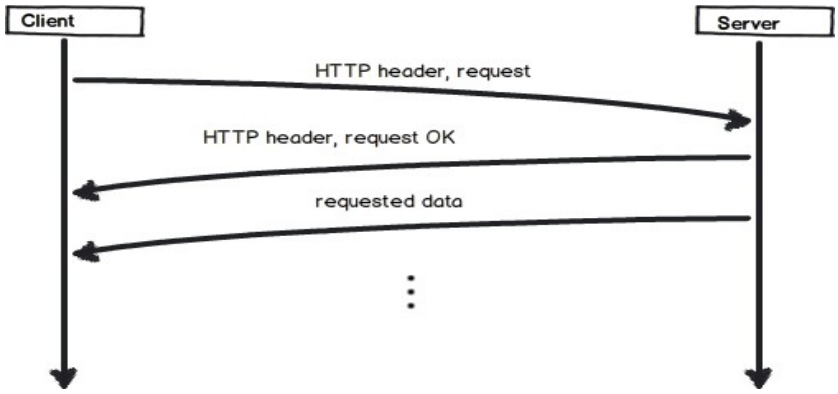
Na obrázku 2.1 je graficky znázorněna komunikace protokolem TCP, ve kterém si účastníci navzájem posílají různé typy packetů: *SYN packet* je používán na indikaci nového požadavku o navázání spojení, *ACK packet* je používán pro potvrzení přijetí požadavku, *FIN packet* je používán k indikaci ukončení spojení.



Obrázek 2.1. Komunikace s pomocí protokolu TCP

Komunikace přes HTTP protokol

HTTP je internetový protokol určený pro přenos informací. Klientem bývá většinou webový prohlížeč. Ten pošle serveru dotaz ve formě čistého textu a čeká na odpověď. Pokud klient pošle další dotaz, bude se jednat o další nezávislý dotaz, tzn. server není jednoduše schopen poznat zda tyto dotazy spolu souvisí. Protokol HTTP je bezestavový, což znamená, že nedokáže uchovávat stav komunikace mezi klientem a serverem. Věškeré informace o stavu komunikace se nacházejí pouze na straně klienta.



Obrázek 2.2. Komunikace pomocí protokolu HTTP

Metoda	**Popis**
GET	Výchozí metoda posílání požadavků, případná data se posílají přímo v URL adrese požadavku.
HEAD	Obdobné jako GET s tím rozdílem, že zde nejsou posílána žádná data.
POST	Metoda pro posílání dat na server, používající se např. při odesílání formulářů.
PUT	Slouží pro ukládání dat na server.
DELETE	Metoda odstraňující data z určité URL adresy.
OPTIONS	Vrací seznam serverem podporovaných HTTP metod.
TRACE	Metoda vracející celou síťovou cestu od klienta k serveru a naopak.

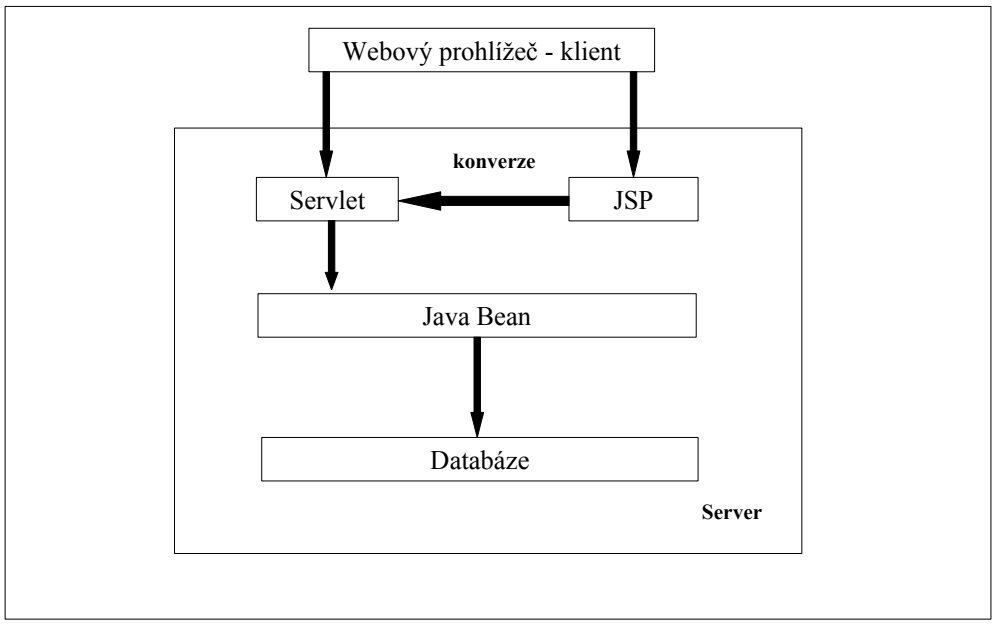
Tabulka 2.1. Metody HTTP protokolu

Pro komunikaci přes protokol HTTP se používá *Java platform, Enterprise Edition* (J2EE), obsahující tzv. *servlety*, obslužné třídy na straně serveru. *Servlety* jsou mapovány společně s URL v souboru *web.xml. Servlet container* je zodpovědný za správné přiřazení požadavku z nějaké URL odpovídajícímu servletu.

J2EE dále obsahuje *Java Server Pages* (JSP), které jsou používány jako vrstva pro zobrazování dynamického obsahu. *JSP* obsahují HTML kód a formátování. *JSP* jsou k URL mapovány odlišně než *servlety* - URL pro dané JSP není definováno v souboru *web.xml*, je totiž odvozeno od adresářové struktury na webu.

Pokud je požadována nějaká URL adresa odpovídající *JSP* souboru, je tento soubor načten do paměti a konvertován na *servlet*. Dále se pro tuto URL adresu používá nově vygenerovaný *servlet*. To znamená, že první načítání URL trvá obvykle déle než všechna následující.

J2EE využívá tzv. *Java Beans* jako třídy pro získávání dat, např. práce s databází. Tyto třídy mohou být používány v *servletech* i v *JSP* souborech.



Obrázek 2.3. Komunikace s použitím J2EE

2.2 C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft. Je založen na jazyce C++ a je možné ho použít k vytváření formulářových, webových a databázových aplikací jak pro desktopy, tak rovněž pro mobilní zařízení. Pro vývoj aplikací se, společně s jazykem C#, velmi často používá *.NET framework* - softwarová komponenta poskytující nástroje a knihovny pro rychlejší a snadnější vývoj aplikací pro operační systém Windows.

2.2.1 Výhody programovacího jazyka C#

- Moderní vysokoúrovňový objektově orientovaný jazyk,
- časové náklady na vývoj aplikací jsou nižší než např. v jazyce C,
- podobně jako Java poskytuje Rapid Application Development (RAD) prostředí – formuláře a vzhled oken lze jednoduše „naklikat“,
- používá *Garbage Collector* pro správu paměti.

2.2.2 Nevýhody programovacího jazyka C#

- Méně dostupných open-source komponent než např. jazyk Java,
- menší optimalizovanost běhových knihoven ve srovnání s Javou.

2.2.3 Programování síťových aplikací v jazyce C#

Stejně jako v Javě existuji i v jazyce C# dvě možnosti pro tvorbu síťových aplikací:

Komunikace přes sockety a protokoly TCP, UDP

Samotná komunikace prostřednictvím socketů a TCP/UDP protokolů byla již popsána v kapitole 2.1.3. Pro zjednodušení tvorby síťových aplikací v jazyce C# poskytuje Microsoft zjednodušenou množinu tříd, nacházejících se ve *jmenném prostoru* (*namespace*) `System.net.sockets`, které umožňují vytvářet plnohodnotné aplikace: [9]

- Třída ***TcpClient*** – byla navržena pro zjednodušené programování aplikací typu TCP klient,
- třída ***TcpListener*** – byla navržena pro zjednodušení psaní serverových aplikací,
- třída ***UdpClient*** – navržena pro zjednodušení vývoje klientů nad protokolem UDP.

Komunikace přes HTTP protokol

Komunikace přes HTTP protokol je popsána v kapitole 2.1.3. V tomto případě je pro tvorbu webových aplikací použit *framework .NET* a jeho součást *ASP.NET*, který je nástupcem technologie *ASP* a přímým konkurentem *JSP*.

ASP.NET řeší problém bezestavovosti protokolu HTTP použitím následujících technik:

- ***ViewState*** – uchovává informace v zakódovaném tvaru ve skrytých formulářových prvcích. Nevýhodou této metody je, že se mezi klientem a serverem musí přenášet větší objem dat.
- ***SessionState*** – ukládá informace na straně serveru, většinou pomocí *cookies*[1], a předává pouze unikátní identifikátor. Zmenší sice objem přenášených dat, ale potřebuje větší výkon na serveru.

2.3 Python

Python je objektově orientovaný programovací jazyk, který navrhl Guido van Rossum v roce 1991. Nabízí významnou podporu integrace s ostatními programovacími jazyky a nástroji a přichází s mnoha doprovodnými knihovnami. [14] Python je vyvíjen jako *open-source,* nabízející instalační balíky pro většinu běžných platforem. Ve většině distribucí systému Linux je Python součástí základní instalace. [15]

2.3.1 Výhody programovacího jazyka Python

- Široká komunita, dobrá podpora,
- rychlý vývoj aplikací,
- znovupožitelnost díky implementovaným modulům a balíkům,
- objektově orientovaný,
- přenositelnost.

1 Cookies jsou data ukládaná na počítači uživatele.

2.3.2 Nevýhody programovacího jazyka Python

- Je plně interpretovaný, tj. je překládán až za běhu programu, a je tedy podstatně pomalejší ve srovnání např. s Javou nebo C,
- chybí mu některé základní konstrukce, např. switch, do-while,
- není standardně zpracováván v HTML,
- chybí mu výchozí hodnoty argumentů funkcí.

2.3.3 Programování síťových aplikací v jazyce Python

Python nabízí dvě úrovně síťového programování. Na nižší úrovni lze přistupovat k základní podpoře socketů, která dovoluje implementovat aplikace typu klient i aplikace typu server pro spojově orientované (TCP) i nespojově orientované (UDP) protokoly.

Python také obsahuje knihovny umožňující přístup k vysokoúrovňovým síťovým protokolům jako jsou FTP, HTTP, apod. [16]

Komunikace přes sockety a protokoly TCP, UDP

Komunikace přes tyto sockety je popsána v odstavci 2.1.3. Pro inicializaci socketu v Pythonu je používána metoda *server.socket(),* dostupná v modulu *socket*. Inicializace socketu vypadá následovně:

```
s = socket.socket (socket_family, socket_type, protocol=0) .
```

Metoda	**Popis**
s.bind()	Metoda pro provázání adresy se socketem, používá ji server.
s.listen()	Tato metoda spustí naslouchání na serveru.
s.accept()	Přijme spojení, čeká na potvrzení.
s.connect()	Tato funkce zažádá o spojení se serverem. Používá ji klient.
s.recv()	Metoda pro přijmutí TCP zprávy.
s.send()	Metoda pro odeslání TCP zprávy.
s.recvfrom()	Metoda pro přijmutí UDP zprávy.
s.sendto()	Metoda pro odeslání UDP zprávy.
s.close()	Tato metoda uzavře socket.
socket.gethostname()	Tato funkce vrátí identifikátor zařízení.

Tabulka 2.2. Metody pro síťovou komunikaci v Pythonu [16]

Komunikace přes HTTP protokol

Funkce HTTP protokolu je popsána v kapitole 2.1.3. Python je často používán jako skriptovací jazyk pro webové aplikace. Použitím *mod_wsgi* se docílí běhu na webovém serveru *Apache* starajícím se o chod webových aplikací. Mod_wsgi je modul HTTP serveru Apache, který zajistí správný chod webových aplikací napsaných v Pythonu. Pro tvorbu těchto aplikací je možné použít různé frameworky, např. *Django, Pylons, TurboGears* atd.

2.4 PHP

PHP je široce používaný skriptovací programovací jazyk používaný pro tvorbu dynamických webových aplikací. Jeho kompilovanou formu lze použít i k tvorbě desktopových aplikací. Syntaxe tohoto jazyka je odvozena od syntaxe jazyků Perl, C apod.

2.4.1 Výhody jazyka PHP

- Funguje na mnoha různých operačních systémech,
- rozsáhlý soubor funkcí v základní knihovně,
- rychlý vývoj aplikací,
- objektově orientovaný,
- rozsáhlá dokumentace v mnoha jazycích,
- široká komunita, mnoho diskuzních fór, skupin zabývajících se PHP,
- syntaxe se podobá jazykům Java, Perl, C apod,
- velká podpora na hostingových službách.

2.4.2 Nevýhody jazyka PHP

- Je interpretovaný a tudíž pomalejší ve srovnání např. s Javou nebo C,
- není tzv. „case-sensitive“, tj. nezaleží na velikosti písmen.

2.4.3 Programování síťových aplikací v jazyce PHP

PHP je známo především jako jazyk vhodný pro tvorbu webových aplikací. V PHP je ovšem možno programovat také síťové aplikace fungující na protokolech TCP/IP.

Komunikace přes sockety a protokoly TCP, UDP

Princip této komunikace je popsán v kapitole 2.1.3. Jazyk PHP umožňuje využít obsáhlou množinu funkcí pro socketové programování. *API (Application Programming Interface)* pro socketové programování obsahuje vše, co je potřeba pro klient-server komunikaci přes TCP/IP, a lze v něm rychle vytvářet jednoduché klient-server aplikace. Hlavní metody tohoto API jsou zobrazené v tabulce 2.3. Množina funkcí pro socketové programování ovšem není standardně povolena. Pro aktivaci této množiny je zapotřebí opětovná kompilace PHP s použitím parametru `– enable- sockets`.

Metoda	**Popis metody**
socket_create	Metoda pro vytvoření socketu.
socket_bind	Tato metoda propojí adresu se socketem.
socket_accept	Přijme spojení.
socket_connect	Tato funkce zažádá o spojení se serverem.
socket_recv	Přijme data z připojeného socketu.
socket_send	Pošle data na připojený socket.
socket_sendto	Pošle data na socket, používá se pro UDP.
socket_recvfrom	Přijme data ze socketu, používá se pro UDP.
socket_close	Tato metoda uzavře socket.

Tabulka 2.3. Metody pro síťovou komunikaci v PHP [17]

Komunikace přes HTTP protokol

Komunikace přes HTTP protokol je popsána v sekci 2.1.3. Jazyk PHP je široce používán pro tvorbu dynamických a interaktivních webových aplikací. Umí pracovat s databázemi, má integrovaný XML parser, často se používá ve spojení s různými šablonovacími systémy, např. *Smarty, FastTemplates* atd. Pro PHP bylo vyvinuto mnoho frameworků, usnadňujících vytváření rozsáhlých webových aplikací. Mezi nejznámější PHP frameworky patří *Drupal, Nette Framework, Zend Framewrok* a mnoho dalších.

3 Metody generování neinformovaných dialogů

V tomto odstavci budou popsány a srovnány tři alternativy pro reprezentaci a vedeni neinformovaných dialogů – AIML, RiveScript a Aerolito.

3.1 AIML

Artificial Intelligence Markup Language (zkráceně AIML) je značkovací jazyk pro umělou inteligenci a XML dialekt pro vytváření chatovacích robotů, konverzujících v přirozeném jazyce. AIML byl vyvinut Dr. Richardem S. Wallacem a volnou komunitou *Alicebot* v letech 1995 až 2000. Původně nevycházel z XML gramatik, ale později byl upraven. Jazyk AIML vytvořil základní pravidla pro první *Alicebot*, *A.L.I.C.E. (Artifical Linguistic Internet Computer Entity)*. Od počátku byl přijat jako standard společností *A.L.I.C.E. AI Foundation*, která vlastní autorská práva.

AIML popisuje skupinu datových objektů nazývaných AIML objekty a chování počítačových programů, které AIML zpracovávají. Protože AIML dokument odpovídá XML struktuře, mohou objekty AIML být také obsaženy uvnitř XML dokumentu.

Objekty v AIML jsou tvořeny částmi zvanými *Topic* a *Category*, které obsahují data parsovaná i neparsovaná. *Parsovaná data* jsou data definující AIML elementy, proměnné a jejich hodnoty. *Neparsovaná data* definují obsah elementů, zpracovaný až na výstupu robota. Elementy zapouzdřují znalostní data typu akce-reakce.

Cíle jazyka:

1. intuitivnost a jednoduchost použití,
2. poskytnutí algoritmu pro vedení neinformovaného dialogu,
3. kompatibilita s XML,
4. jednoduchost psaní programů zpracovávajících AIML dokumenty.

Konstanty a proměnné

Podobně jako v ostatních jazycích a prostředích vyjadřují *konstanty* i v AIML informace, které jsou neměnné. V prostředí konverzace může být jako neměnná informace označeno např. pohlaví konverzačního agenta nebo všeobecně známá fakta (hlavním městem České

republiky je Praha).

V rámci AIML je jedinou povinnou konstantou `name`, která udává název robota, např. řádek č. 3 v příkladu 3.1. V AIML jsou konstanty definovány pomocí elementu *property*, např. řádky 3-6 v příkladu 3.1.

```
1.  <bots>
2.      <bot id="TestBot" enabled="true">
3.          <property name="name" value="iAiml">
4.          <property name="gender" value="male">
5.          <property name="master" value="master">
6.          <property name="birthday" value="2012">
```

Příklad 3.1. Zápis konstant

Protipólem konstant jsou *proměnné* – informace, které se během konverzace mohou měnit, např. jméno uživatele. Proměnné mohou být definovány *staticky*, tj. seznam proměnných a hodnot je vytvořen před zahájením konverzace, nebo *dynamicky*, není-li jméno proměnné známo předem (např. pokud uživatel sdělí název svého oblíbeného filmu). Proměnné jsou v AIML definovány zápisem `<set name="název proměnné">hodnota</set>`. Na řádku č. 4 v příkladu 3.2 je ukázán zápis statické proměnné – její hodnota je známa již při načítání dat. Použití *dynamické proměnné* ukazuje řádek č. 5 v příkladu 3.3.

```
1.  <category>
2.      <pattern>WHO IS MONICA *</pattern>
3.      <template>
4.          <set name ="she">Monica</set>
5.          is a friend of Bill.
6.      </template>
7.  </category>
```

Příklad 3.2. Definování statických proměnných v AIML

```
1.  <category>
2.      <pattern>THE PICTURE *</pattern>
3.      <template>
4.          Do you like
5.          <set name="it"> <star/> </set>?
6.      </template>
7.  </category>
```

Příklad 3.3. Definování dynamických proměnných v AIML

Elementy

AIML elementy se svou strukturou neliší od XML a řídí se XML konvencí. Podle konvence o pojmenovávání elementů a atributů jsou názvy psány *malými písmeny* a pokud jsou víceslovné, oddělují se slova *pomlčkami* (tj. nikoliv „podtržítky“, *underscores,* jak je

běžné v jiných jazycích). Dále je uveden seznam *nejčastěji používaných* AIML elementů, jejich význam a použití. [3]

- **Elementy *category, pattern* a *template* –** element category popisuje jednu možnou reakci (promluvu) agenta na jednu z možných akcí (promluv) uživatele. Reakce je popsána elementem pattern, akce elementem template, viz. níže. Oba elementy jsou bezprostředními potomky category a oba se v něm mohou vyskytnout právě jednou (element template může obsahovat více možných reakcí zapouzdřených v elementu random – v takovém případě algoritmus vedení dialogu náhodně vybere jednu z nich), viz. obrázek 3.4 na řádcích 4-8.

- **Elementy *person, person2 a gender*** – element person udává nahrazení slova v první osobě z obsahu tohoto elementu gramaticky odpovídajícím slovem v třetí osobě a naopak. Element person udává nahrazení slova v první osobě z obsahu tohoto elementu gramaticky odpovídajícím slovem ve druhé osobě a naopak. Elementem gender je definováno nahrazení slova z obsahu tohoto elementu v mužském rodě gramaticky odpovídajícím slovem v ženském rodě a naopak. Definice „gramaticky odpovídajících“ slov je ponechána na implementaci.

- **Element *star*** vrací obsah určité *wildcard* (*zástupného znaku,* většinou znak „*“) použité v otázce. Má nepovinný atribut *index*, který udává jaká *wildcard* má být použita. Nejnižší přijatelná hodnota atributu *index* je 1 (vrátí se obsah poslední použité *wildcard*) a nejvyšší přijatelná hodnota je rovna počtu *wildcard* v elementu *pattern*. Pokud atribut *index* není uveden, doplní se automaticky hodnota 1. Na řádce č. 5 v příkladu 3.4 je uvedeno použití elementu *star* s nedefinovaným atributem index. Atribut *index* je tedy automaticky nastaven na hodnotu 1, která určuje vložení hodnoty poslední hvězdičky použité ve vzoru (akci). V případě konverzace uvedené v příkladu 3.5 se element *star* nahradí hodnotou *Standa*.

```
1.      <category>
2.              <pattern>MY NAME IS *</pattern>
3.              <template>
4.                      <random>
5.                              <li>Hi <star />, how are you?</li>
6.                              <li>Nice to meet you.</li>
7.                              <li>Hello <star />!</li>
8.                      </random>
9.              </template>
10.     </category>
```

Příklad 3.4. Použití elementů *star* a *random* v AIML

*uživatel: My name is **Standa**.*
(Jmenuji se Standa.)
*robot: Hi **Standa**, how are you?*
(Ahoj Stando, jak se mas?)

Příklad 3.5. Ukázka konverzace dle promluvy z příkladu 3.4.

Na řádce č. 3 v příkladu 3.6. je uvedeno vícenásobné použití elementu *star*. Atribut index je zde definován, abychom mohli rozlišit vkládané hodnoty. V případě konverzace uvedené v příkladu 3.7 se element *star* s indexem 1 nahradí hodnotou *red* a element *star* s indexem 2 nahradí hodnotou *color*.

```
1.      <category>
2.              <pattern>MY FAVOURITE * IS *</pattern>
3.              <template>
4.                Really is <star index="1"/> your favourite <star index="2" />?
5.              </template>
6.      </category>
```

Příklad 3.6. Vícenásobné použití elementu *star* v AIML

*uživatel: My favourite **color** is **red**.*
(Má oblíbená barva je červená.)
*robot: Really is **red** your favourite **color**?*
(Opravdu je červená tvá oblíbená barva?)

Příklad 3.7. Ukázka konverzace dle promluvy z příkladu 3.6.

- **Element input** udává, že má být nahrazen kompletním předchozím uživatelským vstupem. Má nepovinný atribut index, který určuje, jaký předchozí uživatelský vstup bude vrácen, řazeno od nejnovějšího po nejstarší. Pokud není atribut uveden, doplní se hodnota 1, odpovídající poslednímu uživatelskému vstupu. Na řádce č. 5 v příkladu 3.8. je zobrazeno použití elementu *input* s nedefinovaným atributem index. Atribut *index* je tedy automaticky nastaven na hodnotu 1, která určuje vložení předposledního uživatelského vstupu (*akce*). V případě konverzace uvedené v příkladu 3.9 se element input nahradí větou *Tony has a new girlfriend* a element star se nahradí dle výše zmíněného postupu. Element *that* na řádce č. 3 v příkladu 3.8 je upřesněním ,že daná promluva se použije pouze v případě, je-li předchozí reakce rovna obsahu tohoto elementu.

```
1.  <category>
2.      <pattern>*</pattern>
3.      <that>WHO TOLD YOU THAT</THAT>
4.      <template>
5.        <star /> said that <input />?
6.      </template>
7.  </category>
```

Příklad 3.8. Použití elementu *input* v AIML

uživatel: ***Tony has a new girlfriend****.*
(Tony má novou dívku.)
robot: Who told you that?
(Kdo ti to řekl?)
uživatel: ***Martin****.*
(Martin.)
robot: ***Martin*** *said that* ***Tony has a new girlfriend****?*
(Martin říkal, že má Tony novou dívku?)

Příklad 3.9. Ukázka konverzace dle promluvy z příkladu 3.8.

- **Element *srai*** určuje, že obsah tohoto elementu bude zpracován jako uživatelský vstup. Odkazuje na jiný element *category*, který určuje pravidlo pro zpracování uživatelského vstupu. Tento element nemá žádné atributy, je ho také možno zapsat pomocí atomického[2] elementu *sr*.

2 Atomický element je takový element, který nemá obsah a není párový. Většinou se jedná o zkrácený zápis jiného elementu.

Specifikace AIML obsahuje ještě mnoho dalších elementů pro práci s proměnnými, formátování textů, apod. Popis všech elementů je k dispozici v oficiálním manuálu AIML. [3]

3.2 RiveScript

RiveScript je scriptovací jazyk pro vytváření chatovacích robotů a jiných konverzačních entit. RiveScript byl původně vytvořen jako modul do programovacího jazyka Perl s názvem *Chatbot::Alpha*. Vyvinul ho Casey Kirsle v roce 2004 jako alternativu k AIML pro jazyk Perl. Oproti AIML, které je postavené na bázi XML, je RiveScript postaven na bázi prostého textu. [19] Nevýhodou RiveScriptu je velmi malá komunita uživatelů a málo zdrojů informací.

RiveScript zvládá:

- Používat regulární výrazy (včetně tzv. *wildcard – zástupný znak, např.* *),
- náhodné odpovědi, přímé odpovědi, přesměrovávání odpovědi,
- formátování textu, nahrazování textu, zpracovávání proměnných,
- používání polí a alternativních slov.

Promluva

V jazyku RiveScript se pro promluvu nepoužívá elementů *category*, *pattern* nebo *template*. V tomto jazyce je promluva dána znaky „+" (*plus pro akci*) a „-" (*minus pro reakci*), viz. příklad 3.10.

```
1.    + How are you?
2.    - I am fine, thank you
```

Příklad 3.10. Promluva v jazyce RiveScript

Konstanty a proměnné

Jazyk RiveScript umožňuje zápis konstant, podobně jako v jazyce AIML, tyto konstanty označují neměnné informace o robotech, např. název robota apod. V příkladu 3.11 je uvedena definice konstant na řádcích 3-5.

```
1.      ! version = 2.0
2.
3.      ! var name = RSBot
4.      ! var age  = 5
5.      ! var gender  = male
```

Příklad 3.11. Zápis konstant v jazyce RiveScript

V jazyce RiveScript jsou specifikovány i informace, které se během konverzace mohou měnit, tzv. proměnnné. Jsou definovány zápisem `<set name=value>`. Operace uložení hodnoty proměnné zde narozdíl od AIML nemá žádnou návratovou hodnotu. Hodnotu proměnné získáme pomocí zápisu `<get name>`. Na řádku č. 2 v příkladu 3.12 je ukázka práce s proměnnými.

```
1.      + my name is *
2.      - <set name=<star>>Nice to meet you, <get name>.
```

Příklad 3.12. Definování proměnných v jazyce RiveScript

*uživatel: My name is **Standa**.*
(Jmenuji se Standa.)
*robot: Nice to meet you, **Standa***
(Těší mne Standa)

Příklad 3.13. Ukázka konverzace dle promluvy z příkladu 3.12.

Používané tagy

Podle konvence jazyka RiveScript mají být názvy tagů psány *malými písmeny*. Dále je uveden seznam *nejčastěji používaných* tagů, jejich význam a použití.

- **Tag *star*** se používá pro získání hodnoty ze zástupného znaku *„*“ (hvězdička)* v uživatelském vstupu. Pokud je ve vstupu více těchto zástupných znaků, přistupuje se k jejich hodnotám pomocí tzv. *indexů*. Tyto indexy se píší hned za slovo star a označují pořadí hodnot, řazeno od poslední použité po první použitou hodnotu, např. tag `<star2>` označuje hodnotu předposledního zástupného znaku, viz. příklad 3.14. Pokud index není definován, uvažuje se hodnota 1.

```
1.      + my favourite * is *
2.      - Really is <star1> your favourite <star2>?
```

Příklad 3.14. Použití tagu *star* v jazyce RiveScript

uživatel: My favourite color is red.
(Má oblíbená barva je červená.) 、
robot: Really is red your favourite color?
(Skutečně je červená tvá oblíbená barva?)

Příklad 3.15. Ukázka konverzace dle promluvy z příkladu 3.14.

- **Tag** *random* definuje náhodný výběr jedné z několika odpovědí. Toho je docíleno pomocí tagu `{random}`. Za tímto tagem je uveden seznam odpovědí, ty začínají znakem *„^“ (stříška)* a jsou oddělené znakem *„|“ (pipe, resp. svislítko)*. Seznam odpovědí je ukončen tagem `{/random}`.

```
1.      + my name is * {random}
2.      ^ Nice to meet you, <star>. |
3.      ^ Hello <star>. |
4.      ^ Hi <star>, how are you? {/random}
```

Příklad 3.16. Náhodná odpověď v jazyce RiveScript

- **Tag *input*** se používá pro získání předchozích uživatelských vstupů. Tento tag může obsahovat *index*, zapsaný hned za slovem *input,* označující pořadí uživatelských vstupů, řazeno od naposledy zadaného po první vstup, např. tag `<input1>`, označuje předchozí uživatelský vstup, viz. příklad 3.17. Pokud *index* není definován, uvažuje se hodnota 1. Znak *„%“ (procento)* na řádce č. 2 v příkladu 3.17 je upřesněním vzoru určující, že danná promluva se použije pouze v případě, je-li předchozí reakce rovna řetězci následujícím za tímto znakem.

```
1.      + *
2.      % who told you that?
3.      - <star> said that <input1>?
```

Příklad 3.17. Použití tagu *input* v jazyce RiveScript

```
uživatel: Tony has a new car.
          (Tony má nové auto.)
robot: Who told you that?
       (Kdo ti to řekl?)
uživatel: Martin.
          (Martin.)
robot: Martin said that Tony has a new car?
       (Martin říkal, že má Tony nové auto?)
```

Příklad 3.18. Ukázka konverzace dle promluvy z příkladu 3.17.

Uvedený seznam obsahuje jen nejčastěji používané tagy jazyka RiveScript. Kompletní seznam všech tagů je k dispozici na oficiální webové prezentaci. [19]

3.3 Aerolito

Aerolito je značkovací jazyk, odvozený od jazyka *YAML ain´t Markup Language (YAML),* snažící se potlačit nevýhody formátování znalostí v jazicích AIML a RiveScript - tj. snaží se být intuitivnější než jazyk AIML a zároveň se pokouší eliminovat nutnost popisovat znalosti vlastním formátem. Je vhodný pro programování chatovacích robotů v jazyce Python. YAML je zde použit pro svou jednoduchost při psaní a čtení kódu. Jednoduše definuje datové typy Pythonu jako seznam, řetězec a slovník. [18]

```
1.  #komentář
2.  key:
3.  - toto je položka seznamu
4.  - 123                          #integer
5.  - [1, 2, 3, 4]                 #seznam
6.  - {1:2, 3:4}                   #slovník
7.  - řetězec                      #řetězec
```

Příklad 3.1. Ukázka YAML souboru

Zdrojová data pro chatovacího robota jsou rozdělena do dvou typů souborů:

- Konverzační soubory obsahující data typu akce-reakce,
- konfigurační soubory, kde jsou uloženy globální proměnné a specifikovány konverzační soubory.

V příkladu 3.2 je na řádcích 1-5 vidět definice globálních proměnných a jejich hodnot. Na dalších řádcích jsou poté uvedené soubory potřebné pro generování dialogů s uživateli. Na následujícím příkladu 3.3 je vidět ukázka konverzačního souboru.

```
1.  version         : v0.1
2.  botname         : aerobot
3.  promenna1       : hodnota1
4.  promenna2       : hodnota2
5.  conversations:
6.          - conv/file1.yml
7.          - conv/file2.yml
8.  synonyms:
9.          - syns/file1.yml
```

Příklad 3.2. Ukázka konfiguračního souboru

```
1.  patterns:
2.          - in:
3.                  - Hello
4.                  - Hi there
5.          out:
6.                  - Hi!
```

Příklad 3.3. Ukázka konverzačního souboru

3.4 Porovnání

Všechny zmíněné jazyky plní dobře svou funkci a hlavní rozdíl je pouze v zápisu kódu. Srovnání mezi výše uvedenými jazyky je proto provedeno především s ohledem na jejich způsob zápisu znalostí. V tomto ohledu má AIML výhodu ve struktuře odvozené z XML. Takto uložená data jsou pro většinu programovacích jazyků lehce zpracovatelná. RiveScript se hodí pro vývoj v programovacím jazyku Perl, zatímco Aerolito se hodí pro vývoj v programovacím jazyku Python. Aerolito i RiveScript jsou experimentálními projekty na vytvoření alternativy k jazyku AIML. V praxi se však používá převážně jazyk AIML.

AIML	RiveScript	Aerolito
<aiml version="X">	! version = X	version: X
<topic name="X">	> topic X	
<category>		
<pattern> HELLO BOT </pattern>	+ hello bot	in: hello bot
<template> Hello human! </template>	- Hello human!	out: Hello human!
<that>XXX</that>	% xxx	after: xxx
<star index="N"/>	<star>, <starN>	<star>, <star 1>, <star N>
<that index="N,M">	<replyN>	
<get name="XXX" />	<get XXX>	<XXX>
<condition name="X" value="Y">	* X eq Y	when: equal: [x, y]
<srai>XXX</srai>, <sr/>	@ xxx, {@XXX}, <@>	out: (rec|XXX)

Tabulka 3.4. Rozdílný zápis kódu

4 Chatovací robot

Chatovací robot je aplikace, která komunikuje s uživateli jejich přirozeným jazykem a umí reagovat na uživatelem zadané otázky formou akcí a reakcí, tj. neinformovaným rozvíjením dialogu. Tato komunikace je realizována na základě souboru vzorů a šablon. Podle zpracovaných akcí je porovnávacím algoritmem vybrána nejlépe odpovídající reakce robota. Obsah této reakce je poté vypsán uživateli jako odpověď.

4.1 Eliza

První chatovací robot byl představen roku 1966 německo-americkým profesorem Josephem Wiezenbaumem a jmenoval se *Eliza*. [20] Tento robot umí komunikovat s uživateli vedením jednoduchých dialogů, pokládáním otázek, zpracováváním proběhlých promluv. Výslednou konverzaci tedy vede v duchu rogerianského terapeuta (Rogerian therapist).

Reakce robota jsou generovány na základě klíčových slov a k nim přiřazeným pravidlům. Tato klíčová slova a pravidla tvoří znalostní bázi robota a nejsou pevně určená, tj. různé instance chatovacího robota mohou mít různé znalostní báze. Znalostní báze robota Eliza existují ve více jazycích, např. velštině, němčině a angličtině. Základní technické problémy, se kterými se Eliza vypořádává jsou např.: [20]

- Zjištění nejdůležitějšího klíčového slova obsaženého v uživatelském vstupu,
- identifikace slov souvicejících s klíčovými slovy, např. pokud je klíčové slovo „You“ (Ty), pak související slovo bývá např. „are“ (jsi),
- výběr vhodných pravidel a generování reakce,
- generování inteligentních reakcí, pokud uživatelský vstup neobsahuje žádné vhodné klíčové slovo,
- zajištění nástrojů pro editaci znalostní báze robota.

uživatel: Men are all alike.
(Muži jsou všichni stejní.)
robot: In what way?
(V jakém smyslu?)
uživatel: They're always bugging us about something or other.
(Oni nás pořád vyslíchají a korigují.)
robot: Can you think of a specific example?
(Můžeš uvést konkrétní příklad?)
uživatel: Well, my boyfriend made me come here.
(Můj přítel mě sem donutil jít.)
robot: Your boyfriend made you come here?
(Tvůj přítel tě sem donutil přijít?)
uživatel: He says I'm depressed much of the time.
(Říká, že jsem pořád deprimovaná.)
robot: I am sorry to hear you are depressed.
(To je mi líto, že jsi deprimovaná.)

Příklad 4.1. Ukázka konverzace s chatovacím robotem Eliza

4.2 Současný stav

V dnešní době existuje spousta chatovacích robotů. Většina těchto robotů je inspirována výše zmíněnou Elizou. Jeden z nejvýznamnějších chatovacích robotů je již jednou zmíněná *ALICE* (Artificial Linguistic Internet Computer Entity) společnosti A.L.I.C.E. AI Foundation. ALICE je robotem komunikujícím v přirozeném jazyce, který několikrát vyhrál Loebnerovu cenu – cena pro nejlepšího chatovacího robota. ALICE využívá pro specifikaci znalostní báze jazyk AIML.

V současnosti existuje několik obdobných chatovacích robotů napsaných v nejrůznějších programovacích jazycích, v následujícím seznam je několik open-source projektů:

- **ProgramD** je jedním z nejpoužívanějších volně dostupných robotů, je napsán v jazyce Java.
- **ProgramR** je implementací chatovacího robota založeném na AIML, který je vytvořen v jazyce Ruby.
- **ProgramO** je projekt chatovacího robota postaveného na jazyce AIML, který byl vytvořen pomocí jazyka PHP a využívá databáze MySQL.

4.3 Generování dialogů

Existuje množina promluv (korpus) a aby chatovací robot byl schopen udržet souvislý tok dialogu (tj. náhodně neskákal z věty na větu), jsou tyto promluvy zařazeny do tzv. *témat (topics)*, které jsou vybírány za běhu, tj. během komunikace. Výběr jednotlivých témat nepodléhá žádnému rozpoznávání – jaké téma má být v daném kontextu vybráno jako další je pevně zadáno již při tvorbě korpusu. Příslušnost dané promluvy k nějakému tématu však není podmínkou – nepřiřazené promluvy nevyužívají žádné údaje zadané uživatelem a jedná se v podstatě o bezkontextové obecné promluvy (např. „Jak se máte?“). Obecný popis AIML souborů, obsahující korpus, je uveden v kapitole 3.1.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <aiml version="1.0">
3.      <topic name= "DOG">
4.          <category>
5.              <pattern>YES</pattern>
6.              <that>IS IT A DOG</that>
7.              <template>Is it a Labrador?</template>
8.          </category>
9.      </topic>
10.     <category>
11.         <pattern>I like animals</pattern>
12.         <template>What is your favourite animal?</template>
13.     </category>
14.     <category>
15.         <pattern>I hate snakes</pattern>
16.         <template>What kind of snakes?</template>
17.     </category>
18. </aiml>
```

Příklad 4.2. Ukázka AIML souboru obsahující korpus o třech interakcích

Každé téma obsahuje alespoň jednu promluvu, resp. dvojici *„akce*$_N$*-reakce*$_N$*“*, kde *akce*$_N$ je aktuální vstup uživatele a *reakce*$_N$ je bezprostřední výstup robota. Chatovací robot tedy jako svou N-tou promluvu vysloví danou reakci, je-li splněna vstupní akce.

uživatel - akce*$_N$*: I hate snakes.
(Nesnáším hady.)
robot - reakce*$_N$*: What kind of snakes?
(Jaký druh hadů?)

Příklad 4.3. Ukázka konverzace

Systém „*akce*$_N$*-reakce*$_N$“ lze dále rozšířit na trojici *„reakce*$_{N-1}$*-akce*$_N$*-reakce*$_N$*“*, kde *reakce*$_{N-1}$ je předchozí promluva robota. Tento rozšířený model tedy napomáhá lépe a přesněji udržet konverzaci na daném tématu.

robot - reakce$_{N-1}$: Is it a dog?
(Je to pes?)
uživatel - akce$_N$: Yes.
(Ano.)
robot - reakce$_N$: Is it a Labrador?
(Je to labrador?)

Příklad 4.4. Ukázka konverzace

Jednoduchá *akce*$_N$, resp. složená „*reakce*$_{N-1}$-*akce*$_N$“ jsou souhrně označovány jako *podmínky* (*paths*) [3]

input (that) (topic),

kde *input* je *N*-tý vstup uživatele, akce$_N$, *that* je volitelná reakce$_{N-1}$, a *topic* je volitelné téma, o kterém má být následující konverzace (určeno konverzačním robotem na základě vyslovení *reakce*$_N$). Pokud některá volitelná část podmínky není definována, nahradí se znakem „*“ (hvězdička).

Chatovací robot používá pro hledání odpovědí dva typy těchto podmínek. Prvním typem je podmínka generovaná ze zdrojových souborů - *vzorová podmínka (match path),* druhým typem je podmínka generovaná z uživatelského vstupu - *vstupní podmínka (input path)*. Tyto podmínky se zpracují porovnávacím algoritmem. Jestliže je nalezena shoda, byla úspěšně nalezena odpověď.

Při zpracovávání vstupního souboru, jehož část je zobrazena na obrázku 4.5, se pro první promluvu na řádcích 4-8 vygeneruje vzorová podmínka

IT IS SPORT COUPE (that) REALLY WHAT TYPE OF CAR (topic) CAR.

Protože první interakce obrázku 4.5 (řádky 4-8) je uzavřena v elementu *topic* s názvem *CAR,* je do poslední části vzorové podmínky vloženo slovo CAR. Prostřední část je určená obsahem elementu *that* a první část je určená obsahem elementu *pattern*.

Druhá interakce na obrázku 4.5 se nachází na řádcích 10-13. Tato promluva není uzavřena v elementu topic, ani neobsahuje element that a tedy vzorová podmínka vypadá následnovně:

*I BOUGHT NEW CAR (that) * (topic) *.*

```
1.      <topic name= "CAR">
4.              <category>
5.                      <pattern>It is sport coupe.</pattern>
6.                      <that>REALLY WHAT TYPE OF CAR</that>
7.                      <template>Very good choice.</template>
8.              </category>
9.      </topic>
10.     <category>
11.             <pattern>I bought new car.</pattern>
12.             <template>
13.                     Really, what type of <set name="topic">car</set>?
14.             </template>
15.     </category>
```

Příklad 4.5. Ukázka AIML souboru

Pokud je zadán uživatelský vstup odpovídající druhé podmínce - *„I bought new car.“*, je vygenerována vstupní podmínka

*I BOUGHT NEW CAR (that) * (topic) *.*

Tato podmínka se rovná vzorové podmínce a je tedy zpracována reakce odpovídající druhé interakci. Při zpracování reakce je pomocí elementu `<set name="topic">` nastaveno téma na hodnotu „CAR“ a robot odpoví větou „Really, what type of car?“. Při další akci již je zvoleno téma promluvy a je definována i předchozí reakce robota. Pokud je zadán uživatelský vstup odpovídající první podmínce - *„It is sport coupe.“*, je vygenerována vstupní podmínka

IT IS SPORT COUPE (that) REALLY WHAT TYPE OF CAR (topic) CAR.

Tato podmínka se rovná vzorové podmínce a je tedy zpracována reakce odpovídající první interakci a vrácena odpověď *„Very good choice.“*.

5 Vlastní implementace

Chatovací robot, popisovaný v této kapitole, je mojí vlastní implementací chatovacího prostředí využívající znalostní bázi specifikovanou jazykem AIML. Jako korpusu je zde využito volně dostupných A.L.I.C.E. AIML souborů, které umožňují komunikaci v angličtine. [4] Protože je využíván anglicky psaný korpus, jsou všechny ukázky uváděny v anglickém jazyce.

Tato kapitola popisuje implementovaného chatovacího robota, jednotlivé třídy a nejdůležitější metody použité při vývoji této aplikace. Dále je zde vysvětleno, jak tento robot vede dialog v souladu s odstavcem 4.3, Generování dialogu se dělí na tři části – 1. načítání dat, 2. zpracování uživatelského vstupu a 3. generování odpovědi.

5.1 Zvolené technologie

PHP

Prvním krokem při volbě technologií pro tvorbu chatovacího robota je volba programovacího jazyka, ve kterém se bude tato aplikace vyvíjet. Jako nejlepší možnost z uživatelského hlediska se jeví tvorba chatovacího robota jako webové aplikace. To má několik výhod, mezi které patří např. platformová nezávislost, snadný přístup k aplikaci bez nutnosti instalace klienta. Klientem je v tomto případě kterýkoliv webový prohlížeč. Další vývoj nebo update aplikace se provádí pouze na straně serveru. Pro tvorbu chatovacího robota jako webové aplikace se hodí programovací jazyk PHP, jelikož je určen pro tvorbu webových aplikací, je podporován na většině webových serverech a je multiplatformní.

AIML

Jelikož jazyk PHP má zabudovaný XML parser a proto bude přístup k XML datům rychlý a snadný, hodí se použít jazyk AIML, jakožto zdroj dat pro vedení neinformovaných dialogů. Další výhodou jazyku AIML je, že je nejvíce požívaným jazykem pro tvorbu chatovacích robotů a je dobře dokumentován.

MySQL

Jako úložiště byl zvolen databázový systém MySQL, který je díky své snadné použitelnosti a výkonu jedním z nejpoužívanějších volně dostupných nástrojů pro tvorbu webových aplikací.

5.2 Struktura aplikace

Všechny soubory pro bezproblémový chod aplikace musejí být umístěny na serveru, na kterém běží webový server s podporou PHP a podporou MySQL. Dále musí být nastaveny konstanty ze souboru `./inc/settings.inc.php`, viz. tabulka 5.1.

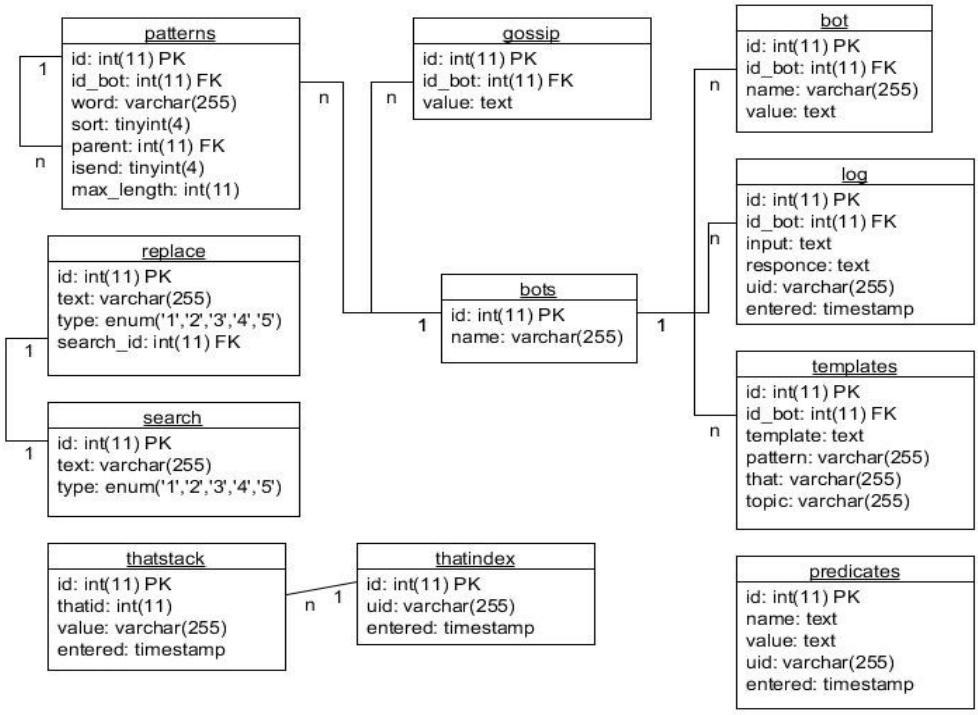
Konstanta	**Popis**
DB_HOST	Název MySQL serveru
DB_USER	Uživatel s přístupem k databázi
DB_PWD	Heslo pro přístup k databázi
DB	Název databáze
HOME_URL	URL adresa aplikace
HOME_DIR	Cesta k aplikaci na webovém serveru

Tabulka 5.1. Konstanty potřebné pro běh aplikace

Jméno adresáře	**Funkce adresáře**
./cls/	V tomto adresáři jsou uloženy třídy potřebné pro chod aplikace.
./css/	Zde se nachází soubory s kaskádovými styly. To jsou soubory používané pro formátování obsahu, například nastavení barvy, velikosti písma, pozadí, zarovnání atd.
./data/	V adresáři ./data/ jsou AIML soubory, které jsou nutné pro generování dialogů s uživatelem. Dále se zde nachází soubor *startup.xml*, ve kterém jsou informace o robotech, konstanty, a informace potřebné pro normalizace textu.
./inc/	V tomto adresáři jsou soubory, ve kterých jsou uloženy obecné

	funkce, nastavení přístupu k databázi, nastavování globálních proměnných apod.
./tpl/	Zde jsou umístěny šablony HTML pro zobrazení obsahu.

Tabulka 5.2. Adresářová struktura aplikace



Obrázek 5.1. ERA model databáze

Tabulka	**Popis tabulky**
bot	Obsahuje informace o robotech - konstanty. Název konstanty určuje sloupec *name*, její hodnotu sloupeček *value* a sloupec *id_bot* určuje, ke kterému robotu se vztahují informace.
bots	Zde jsou uloženy informace o dostupných robotech. Je zde jen informace o jejich názvu (sloupec *name*).
predicates	Zahrnuje proměnné, vytvořené za běhu aplikace. Název proměnné označuje sloupec *name*, hodnotu proměnné určuje sloupec *value*, doba vložení proměnné je ve sloupci *entered* a *id*

	uživatele, se kterým byla konverzace vedena, je ve sloupci *uid*. *Id* je jednoznačný identifikátor uživatele.
gossip	Zde se ukládá obsah z elementů *gossip*. Obsah těchto elementů označuje sloupec *value* a sloupec *id_bot* označuje *id* robota, se kterým je vedena komunikace.
log	V této tabulce je uložena historie komunikace. Sloupec *id_bot* označuje *id* robota, se kterým je vedena komunikace, sloupec *input* označuje uživatelský vstup, sloupec *response* označuje odpověď, *id* uživatele, se kterým byla konverzace vedena, je ve sloupci *uid* a ve sloupci *entered* je čas komunikace.
patterns	Tato tabulka obsahuje informace o vzorových řetězcích. Sloupec *id_bot* označuje *id* robota, kterému vzorový řetězec patří, sloupec *word* označuje část vzorového řetězce, sloupeček *sort* označuje typ části vzorového řetězce, *parent* je *id* předchozí části a *isend* indikuje koncový prvek vzorového řetězce.
search	Tabulka *search* obsahuje data potřebná pro normalizace textu. Sloupec *text* obsahuje řetezec, který bude nahrazen a sloupeček *type* označuje typ normalizace.
replace	Tabulka *replace* obsahuje data potřebná pro normalizace textu. Sloupec *text* obsahuje řetězec, který bude dosazen za původní, sloupec *type* označuje typ normalizace, sloupec *search_id* označuje *id* prvku z tabulky *search*, tj. prvku, který má být nahrazen.
templates	Tato tabulka obsahuje data potřebná pro sestrojení odpovědi. Sloupec *id_bot* označuje *id* robota, kterému odpověď patří, sloupec *template* obsahuje text, ze kterého se sestrojí odpověď, sloupec *pattern* obsahuje vzor, pro který je zpracována danná odpověď, sloupec *that* a *topic* obsahují informaci o elementech *that* a *topic* spojených s dannou odpovědí.
thatindex a thatstack	Tyto tabulky obsahují informace o generovaných odpovědích.

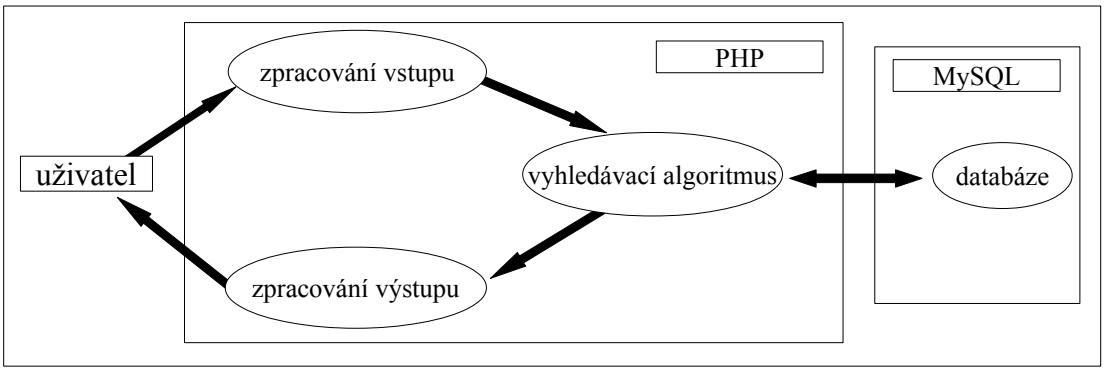
Tabulka 5.3. Databázová struktura aplikace

Třída	Popis třídy
CBotLoader	Tato třída slouží pro ukládání zdrojových dat do databáze.
CGraphicon	Tato třída je implementací vyhledávacího algoritmu.
CModel	Třída CModel je potomkem třídy db a obsahuje funkce pro práci s daty uloženými v databázi.
CReplacer	CReplace slouží pro normalizace typu nahrazení.
CResponce	Tato třída zpracovává uživatelský vstup a vytváří odpověď.
db	Obecná třída pro práci s db, obsahující základní funkce, jako je připojení do databáze, provedení SQL dotazu atd.

Tabulka 5.4. Použité třídy

5.3 Funkční cyklus robota

Po zadání uživatelského vstupu je tento vstup zpracován (viz. kapitola 5.5) na tzv. vstupní podmínku (viz. odstavec 4.3). Poté je pomocí prohledávacího algoritmu hledána tzv. vzorová podmínka rovnající se vstupní podmínce. Nalezená vzorová podmínka ukazuje na příslušnou odpověď robota. Odpověď robota musí být ještě před odesláním uživateli zpracována, jelikož může obsahovat různé řídící elementy (např. element *random* nebo *star*, viz. odstavec 3.1). Zpracovaná odpověď je odeslána uživateli.



Obrázek 5.2. Struktura chatovacího systému

5.4 Načítání dat

Načítání dat se provádí před počátkem konverzace a zpracovaná data se ukládají do databáze. Načítání dat zajišťuje třída `CBotLoader`, využívající třídu `CModel` pro práci s databází a třídu `CReplacer` pro načtení substitucí.[3] Následující soubory jsou užitím těchto tříd, XML parseru a jednoduchých funkcí zpracovány.

Robot je popsán v souboru `startup.xml`, který je načten jako první věc a na jeho základě jsou načteny ostatní součásti celého chatovacího systému. Soubor `startup.xml` je odvozen od konfiguračního souboru aplikace ProgramD, která je jednou z nejpoužívanějších open-source implementací chatovacího systému.

Všechny v systému dostupné roboty jsou popsány v seznamu *bots*, v němž každý je identifikován svým jménem (atribut *id*) a jsou mu přiřazeny určité vlastnosti (elementy *property*). Každý z robotů umí konverzovat jistým způsobem, který je určen elementem *learn* (adresa AIML souboru, resp. hvězdička pro načtení všech AIML souborů v daném adresáři).

```
1.  <bots>
2.      <bot id="TestBot" enabled="true">
3.          <property name="name" value="iAlice">
4.          <property name="gender" value="male">
5.          <property namè="master" value="master">
6.          <property name="language" value="english">
7.          <learn>*</learn>
8.      </bot>
9.  </bots>
```

Příklad 5.1. Definice robota v AIML

Element *substitutions* v příkladu 5.2 obsahuje *substituční* elementy *input*, *gender*, *person*, *person2*. Obsah těchto elementů je načten do databáze a později použit (např. při záměně slov v textu, odstranění nevhodných znaků apod.) při zpracování vstupu uživatele, resp. výstupu robota. Každý ze *substitučních* elementů obsahuje jeden nebo více potomků. Potomky jsou elementy *substitute* a mají atributy *find* resp. r*eplace*, uchovávající řetězec nahrazovaný resp. nahrazující.

3 Substituce je záměna řetězců, potřebná pro provedení upravy textu.

```
1.  <substitutions>
2.      <input>
3.          <substitute find=",)" replace="smile">
4.          <substitute find=";)" replace="smile">
5.          <substitute find=":)" replace="smile">
6.          ...
7.      </input>
8.      <gender>
9.          <substitute find="he" replace="she">
10.         <substitute find="she" replace="he">
11.         <substitute find="his" replace="her">
12.         ...
13.     </gender>
14.     <person>
15.         <substitute find="i was" replace="he was">
16.         <substitute find="he was" replace="i was">
17.         <substitute find="he has" replace="i have">
18.         ...
19.     </person>
20.     <person2>
21.         <substitute find="with you" replace="with me">
22.         <substitute find="with me" replace="with you">
23.         <substitute find="love you" replace="love me">
24.         ...
25.     </person2>
26. </substitutions>
```

Příklad 5.2. Definice substitucí v AIML

Po elementu *substitutions* je v tomto souboru element *sentence-splitters*. Potomky jsou elementy *splitter*, které mají atribut *value*, který obsahuje oddělovač vět.

```
1.  <sentence-splitters>
2.      <splitter value=".">
3.      <splitter value="!">
4.      <splitter value="?">
5.  </sentence-splitters>
```

Příklad 5.3. Definice oddělovačů v AIML

Načítání konverzačního robota

Nejprve je vytvořena instance třídy `CBotLoader` a je nastaven adresář s datovými soubory. Dále je nastaven *XML parser,* kterým jsou zpracovávána data ze souboru `startup.xml`.

V příkladu 5.4. je ukázka části kódu funkce *aimlEndTag*, který zpracovává promluvy z elementu *category*. Řádky 2-5 ořezávají mezery ze začátku a konce proměnných *template*, *pattern*, *that* a *topic*. Na řádcích 7-9 zjistíme, zda jsou definovány všechny části vzorové podmínky. Nejsou-li, jsou doplněny znakem *„*“ (hvězdička)*. Uložení vzorové podmínky spolu s příslušnými odpověďmi je na řádcích 10-16. Zbylé řádky vymažou proměnné *template*, *that* a *pattern,* aby se do nich mohla ukládat data z dalších elementů.

```
1.      case 'CATEGORY':
2.          $this -> template = trim($this -> template);
3.          $this -> topic = trim($this -> topic);
4.          $this -> that = trim($this -> that);
5.          $this -> pattern = trim($this -> pattern);
6.
7.          if ($this -> topic == "") $this -> topic = "*";
8.          if ($this -> that == "") $this -> that = "*";
9.          if ($this -> pattern == "") $this -> pattern = "*";

10.         $path = "<input> " . $this -> pattern . " <that> " . $this -> that . " <topic> " . $this -> topic;
11.         $patternId = $this -> db -> insertSentence($this -> actualBot, $path);
12.     $this -> db -> insertTemplate($this -> actualBot,
13.                                   $patternId, $this -> template,
14.                                   $this -> pattern,
15.                                   $this -> that,
16.                                   $this -> topic);
17.         $this -> template = "";
18.         $this -> that = "";
19.         $this -> pattern = "";
20.     break;
```

Příklad 5.4. Ukázka zpracování elementu *category* funkcí aimlEndTag

Po úspěšném uložení informací o robotech a ostatních součástí systému, které se tyto roboty mají naučit, se nastaví *AIML parser* a začnou se zpracovávat promluvy uvedené v AIML souborech. Vzorové podmínky interakcí uvedených v jednotlivých souborech jsou ukládány do grafu (viz. obrázek 5.3), jehož uzly jsou jednotlivá slova vzorových podmínek. Dále také ukládá do databáze příslušné odpovědi.

5.5 Zpracování uživatelského vstupu

Každý uživatelský vstup je zpracováván pomocí třídy `CResponce`, využívající tříd `CModel` pro práci s databází a `CReplacer` pro nahrazování nevhodných řetězců (uvedeno níže v tomto odstavci). Zpracováváním vstupu je myšlena úprava textu před samotným vyhledáváním odpovědi (tj. před porovnáváním vstupní podmínky se vzorovými podmínkami). Jako první se na uživatelský vstup aplikuje nahrazení nevhodných řetězců, popsané dále. Poté se provede rozdělení vstupu na věty, které se převedou na řetězec použitelný ve vstupní podmínce, viz. kapitola 4.3.

Bez počátečního nahrazení nevhodných řetězců by informace obsažené v uživatelském vstupu mohly při dalším zpracování pozměnit svůj význam.

Funkce nahrazování:

- Ze zkratek typu *Mr.,Dr.* apod se odstraní tečky,
- adresy jako např. *http://alicebot.org* se interpretuje jako *http alicebot dot org*, tečky v příponách souborů jsou odstraněny,
- různé zkrácené formy (aren't, can't, atd.) jsou nahrazeny jejich delšími variantami (are not, cannot),
- u desetinných čísel jsou desetinné tečky nahrazeny slovem point (0.32 → 0 point 32), aby se zamezilo špatnému rozdělení vět,
- hovorové výrazy (např. *wanna, ur*) jsou nahrazeny spisovnými výrazy (např. *want to, you are*),
- znaky {", ;\:&)(-/'} jsou nahrazeny mezerou.

uživatel: ***I can´t understand.***
upravený vstup: ***I cannot understand.***
(Nerozumím)

uživatel: ***That was only joke :-)***
upravený vstup: ***That was only joke***
(Byl to jen vtip)

Příklad 5.5. Ukázka nahrazování nevhodných řetězců

Po prvotním zpracování se upravený vstup rozdělí na věty dle znaků uvedených v elementu *sentence-splitters*, viz. odstavec 5.3. Každá věta se uloží do databáze jako proměnná s názvem *input* a dále je zpracována samostatně.

uživatel: ***I can´t understand. Can you tell me more?***
upravený vstup: ***I cannot understand.***
Can you tell me more?
(Nerozumím, můžeš mi říct více?)

Příklad 5.6. Ukázka rozdělení vět

Dále se každá věta převede na řetězec použitelný ve vstupní podmínce dle následujících pravidel:

- Pokud je znak malé písmeno, je nahrazen velkým písmenem,
- pokud znak není písmeno, je nahrazen mezerou.

uživatel: ***Hi, how are you?***
upravený vstup: ***HI HOW ARE YOU***
(Ahoj, jak se máš?)

uživatel: ***Do you know site http://www.alicebot.org?***
upravený vstup: ***DO YOU KNOW SITE HTTP WWW ALICEBOT ORG***
(Znáš stránky http://www.alicebot.org?)

Příklad 5.7. Převod na řetězec použitelný ve vstupní podmínce

Vstup	**Nahrazování nevhodných řetězců**	**Rozdělení vět**	**Převod na řetězec použitelný ve vstupní podmínce**
"What time is it?"	"What time is it?"	"What time is it"	"WHAT TIME IS IT"
"Quickly, go to http://alicebot.org!"	"Quickly, go to http://alicebot dot org!"	"Quickly, go to http://alicebot dot org"	"QUICKLY GO TO HTTP ALICEBOT DOT ORG"
":-) That's funny."	"That is funny."	"That is funny"	"THAT IS FUNNY"
"I don't know. Do you, or will you, have a robots.txt file?"	"I do not know. Do you, or will you, have a robots dot txt file?"	"I do not know" "Do you, or will you, have a robots dot txt file"	"I DO NOT KNOW" "DO YOU OR WILL YOU HAVE A ROBOTS DOT TXT FILE"

Tabulka 5.5. Příklady zpracování vstupu

Pro vysvětlení je v následující tabulce 5.6. znázorněno zpracování vstupního řetězce „*I dont know Peter. Do you know him?*“ (Neznám Petra, ty ho znáš?).

Vstup	*I don't know Peter. Do you know him?*
Nahrazení nevhodných znaků	*I do not know Peter. Do you know him?*
Rozdělení vět	*I do not know Peter.* *Do you know him?*
Převod na řetězec použitelný ve vstupní podmínce	*I DO NOT KNOW PETER* *DO YOU KNOW HIM*
Vstupní podmínky	*I DO NOT KNOW PETER (that) * (topic) ** *DO YOU KNOW HIM (that) * (topic) **

Tabulka 5.6. Příklad zpracování vstupu

Po zpracování uživatelského vstupu je zavolána funkce *reply* třídy `CResponce`, která načte uložené hodnoty proměnných *that* a *topic.* Výsledek zracování vstupu se použije společně s načtenými hodnotami that a topic k sestavení vstupní podmínky(viz. odstavec 4.3) a vytváří se odpověď postupem popsaným v kapitole 5.5.2.

V následujícím příkladu 5.8. je na řádcích 2-5 ověreno, zda existuje tázaný robot. Pokud neexistuje, vypíše se chyba a funkce skončí s návratovou hodnotou *false*. Řádky č. 8 a 9 slouží k načítání hodnot proměnných *that* a *topic.* Na řádce č. 10 je volána funkce pro zpracování uživatelského vstupu.

```
1.    public function reply($userInput, $uniqueId = ""){
2.        if (!$this -> botId || $this -> botId == NULL) {
3.            print "Undefined bot<br />\n";
4.            return false;
5.        }
6.
7.        $this -> uid = $uniqueId;
8.        $this -> that = $this -> db -> getThat(1,1);
9.        $this -> topic = $this -> db -> getPredicate("TOPIC");
10.       $inputs = $this -> normalize($userInput);
11.       ...
```

Příklad 5.8. Ukázka části kódu funkce *reply* zpracovávající uživatelský vstup

5.6 Generování odpovědi

Po zpracování uživatelského vstupu je zapotřebí nalézt příslušnou odpověď. K tomu slouží třída `CResponce`, využívající tříd `CModel` pro práci s databází, `CReplacer` pro nahrazování řetězců a `CGraphicon` pro nalezení odpovědi. Pokud je z uživatelského vstupu vygenerováno více vstupních podmínek (viz. tabulka 5.6), je pro každou vstupní podmínku nalezena odpověď a tyto nalezené odpovědi se před odeslání uživateli složí do jedné (např. „*odpověď*$_1$ *odpověď*$_2$“).

5.6.1 Prohledávání grafu

Třída `CGraphicon` je implementací procházení grafu do hloubky, které je častou implementací AIML porovnávací funkcionality. Graf je struktura uchovávaná v databázi v tabulce patterns. V této struktuře je vždy jedno slovo vzorového řetězce uloženo jako jeden uzel s odkazem na svého předka a hodnotou délky maximální cesty k listu grafu (koncovému bodu).

Hledáním slova s nejdelší cestu k listu (ke koncovému bodu grafu) je zaručeno, že se vstup bude nejdříve porovnávat s nejpřesnějšími vzory. Algoritmus si také uchovává historii již prohledaných uzlů pro zrychlení vyhledávání.

Pokud kořen grafu obsahuje klíč *„*“ (hvězdička)* a ten odkazuje na listový (koncový) uzel, pak algoritmus vždy najde shodu. V příkladu 5.10. je několik vzorů, které začínají obdobně. Na obrázku 5.3. je znázorněn úsek *grafu* odpovídající příkladu 5.10.

1. Dáno: vstupní podmínka (*COND*); graf obsahující vzorové podmínky; pole prohledaných uzlů grafu (CLOSED), id předchozího slova(PARENT)**.**

2. Dokud *COND* obsahuje nějaké slovo:

   2.1. *WORD* := první slovo nezpracované části *COND*.

   2.2. Hledej v grafu *WORD* s nejdelší cestou (*údaj o délce představuje počet termů od daného slova do listového uzlu grafu*);

      2.2.1. Pokud nalezeno: id nalezeného uzlu ulož do *PARENT* a do *CLOSED*; odeber *WORD* z *COND* a opakuj krok 2.

   2.3. Hledej v grafu *wildcard* s nejdelší cestou;

      2.2.1. Pokud nalezeno: id nalezeného uzlu ulož do *PARENT* a do *CLOSED*; odeber *WORD* z *COND* a opakuj krok 2.

   2.4. Pokud je definováno *PARENT*, vrať se o krok zpět – obnov předchozí stav *PARENT* a *COND* a pokračuj krokem 2.

   2.5. Zastav algoritmus s výsledkem nenalezeno.

3. Pokud je definováno *PARENT*, zastav algoritmus; vrať odpovídající reakci.

4. Zastav algoritmus s výsledkem nenalezeno.

Příklad 5.9. Modifikovaný algoritmus vyhledávání do hloubky pro hledání reakce agenta

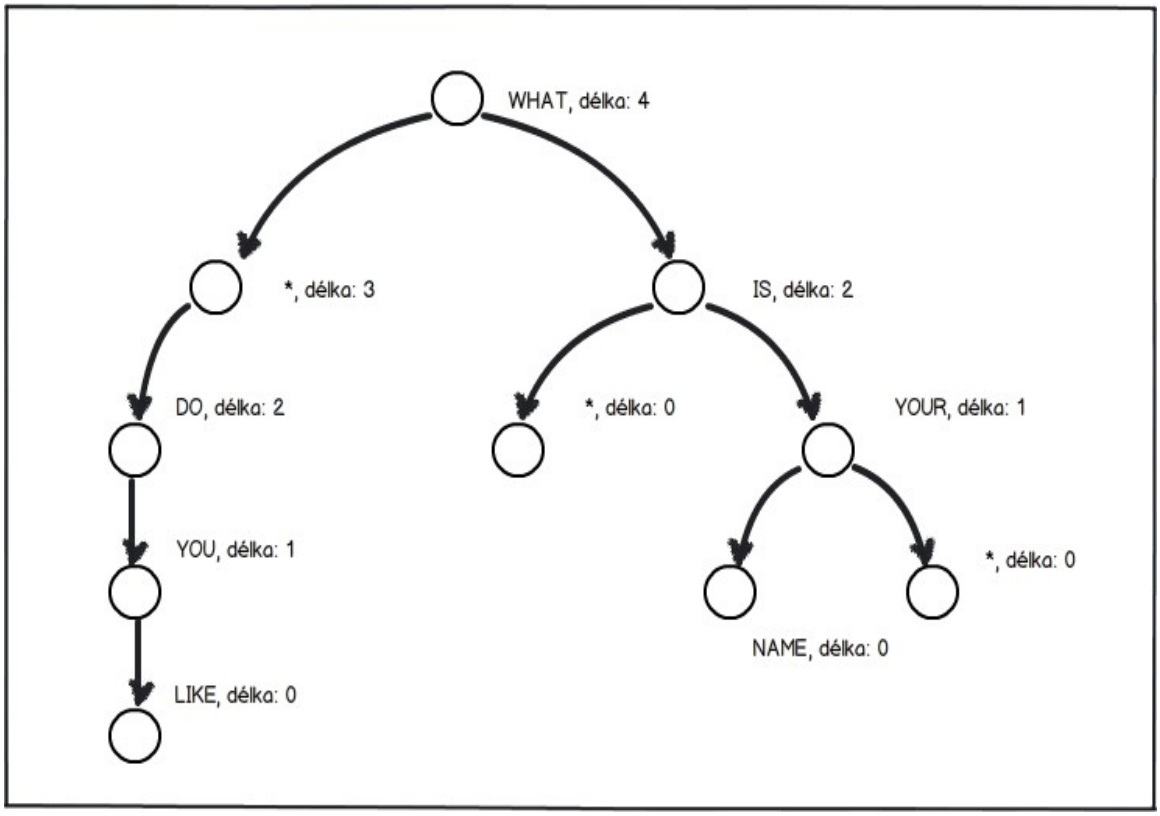
```
1.      <category>
2.              <pattern>WHAT IS *</pattern>
3.              <template>I don't know what <star /> is.</template>
4.      </category>

5.      <category>
6.              <pattern>WHAT IS YOUR *</pattern>
7.              <template>Don't talk about me.</template>
8.      </category>

9.      <category>                              `
10.             <pattern>WHAT * DO YOU LIKE?</pattern>
11.             <template>
12.                     Stop talking about me, what <star /> do you like?
13.             </template>
14.     </category>

15.     <category>
16.             <pattern>WHAT IS YOUR NAME</pattern>
17.             <template>My name is <bot name="name" /></template>
18.     </category>
```

Příklad 5.10. Ukázka elementů *category* s podobně začínajícími vzory



Obrázek 5.3. Ukázka grafu sestrojeného dle příklad 5.10.

Postup vyhledávání při uživatelském vstupu *What color do you like?* :

- Hledané slovo je *WHAT*, toto slovo bylo nalezeno a je odebráno ze vstupu, uzel obsahující slovo *WHAT* je vložen do množiny prohledaných,
- další hledané slovo je *COLOR*, algoritmus vyhledá nejdříve slovo (ne *wildcard*) s nejdelší cestou, tj. *IS*, což neodpovídá, uzel se slovem *IS* je vložen do množiny prohledaných,
- znovu je hledáno slovo *COLOR*, algoritmus nenajde opět uzel se slovem *IS*, jelikož je v množině prohledaných,
- je nalezena *wildcard* „*“ (hvězdička), slovo *COLOR* je odebráno ze vstupu, uzel s touto wildcard je vložen do množiny prohledaných,
- hledáno slovo *DO,* což bylo nalezeno, slovo *DO* je odebráno ze vstupu, uzel se slovem *DO* je vložen do množiny prohledaných,
- hledáno slovo *YOU* a nalezeno, slovo *YOU* je odebráno ze vstupu, uzel se slovem *YOU* je vložen do množiny prohledaných,
- hledáno slovo *LIKE* a nalezeno, slovo *LIKE* je odebráno ze vstupu, uzel se slovem *LIKE* je vložen do množiny prohledaných,
- vstup je prázdný a proto je ukončen **algoritmus a vrácena nalezená odpověď.**

5.6.2 Zpracování odpovědí

Pro zpracování nalezených odpovědí slouží několik funkcí, jejichž účelem je *formátovat výstup, doplnit hodnoty proměnných, vypisovat konstanty, přesměrovávat výstup na hledání další odpovědi, náhodný výběr z více odpovědí* a spoustu dalších funkcí.

Uživatelský vstup	Do you read the Bible? (Čteš Bibli?)
Odpovídající interakce	<category> <pattern>DO YOU READ THE *</pattern> <template> I read <set name="it"> <star/> </set> from time to time. </template> </category>
Nalezená odpověď	I read <set name="it"> <star/> </set> from time to time.
Zpracovaná odpověď	**I read Bible from time to time.** (Čtu Bibli čas od času.)

Tabulka 5.7. Příklad zpracování odpovědi

Pokud se v odpovědi vyskytuje jeden ze substitučních elementů person, person2 nebo gender, je jeho obsah zpracován postupem popsaným v odstavci 3.1. Definice gramaticky odpovídajících slov (tj. seznam slov nahrazovaných a nahrazujících) je dána obsahem elementu substitutions, viz. odstavec 5.4.

Element	**Slovo nahrazované**	**Slovo nahrazující**
Person	He was	I was
Person2	To me	To you
Person2	To you	To me
Gender	He	She
Gender	His	Her
Gender	With her	With him

Tabulka 5.8. Příklad zpracování substitučních elementů

V příkladu 5.11. je vidět funkce *getRandom*, která náhodně vybere odpověď. Pokud neexistuje odpověď, vrátí hodnotu *false*. Na řádce č. 8 se nachází *náhodný výběr* z pole odpovědí. Pokud je tato odpověď nalezena, je vrácena.

```
1.  private function getRandom($elm) {
2.      if (!isset($elm['children']) || !is_array($elm['children'])) return FALSE;
3.
4.      $lis = array();
5.      foreach ($elm['children'] as $k => $v)
6.          if (strtoupper($v['tag']) == "LI") $lis[] = $k;
7.
8.      $rand_keys = array_rand($lis, 1);
9.      return $this -> recurseChildren(
10.         $this -> getChildren($elm['children'][$lis[$rand_keys]]));
11. }
```

Příklad 5.11. Ukázka funkce *getRandom* pro náhodný výběr odpovědi

V příkladu 5.12. je znázorněna funkce *getBotValue*, která vrací hodnotu příslušné konstanty. Pokud element *bot* nemá atributy, je vrácena hodnota *false*. Jinak jsou na řádce č. 3 převedeny atributy na velká písmena pomocí funkce *array2uppercase*. Dále je testováno, zda existuje atribut *name*. Pokud neexistuje, je vrácena hodnota *false*. Pokud existuje, je z databáze načtena hodnota příslušné konstanty, určené atributem *name*.

```
1.  private function getBotValue($elm) {
2.      if (!isset($elm['attributes'])) return FALSE;
3.          $attrs = array2uppercase($elm['attributes']);
4.
5.      if (!$attrs || !isset($attrs['NAME'])) return FALSE;
6.          return $this -> db -> getBotValue($this -> botId, $attrs['NAME']);
7.  }
```

Příklad 5.12. Ukázka funkce *getBotValue* pro výpis konstanty

6 Testování

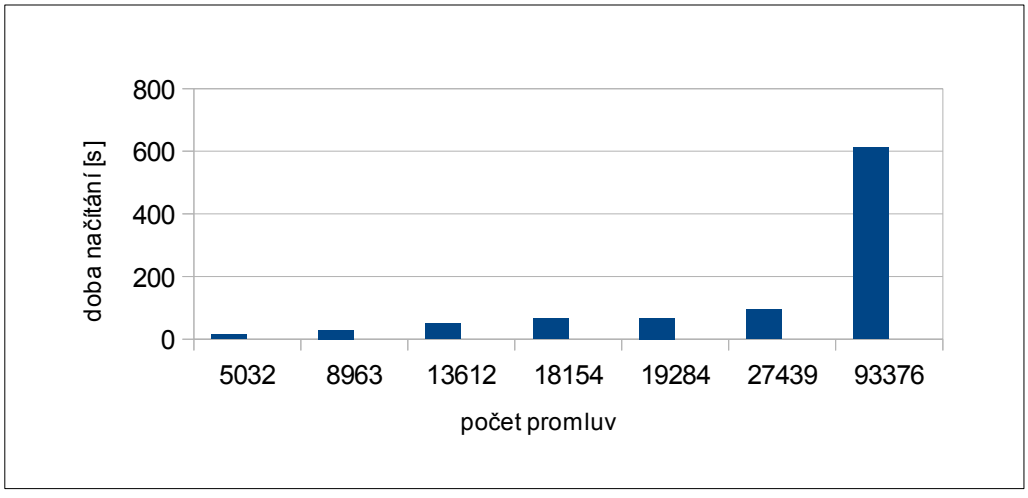
Zpracování vstupních dat

Tato část je věnována rychlosti zpracování vstupních souborů, které zahrnuje rozparsování a následné uložení potřebných informací do databáze. Byla měřena rychlost, počet vygenerovaných uzlů v grafu, počet uložených šablon a výsledná velikost databáze. Pro porovnávání naměřených výsledků je spíše než počet zpracovaných souborů důležitý počet načítaných promluv (předpokládá se, že v souborech se nevyskytují duplicity).

Jak se lze na základě tabulky 6.1 a grafu 6.1 přesvědčit, čas potřebný k načtení vstupních dat je přímo závislý na velikosti vstupních dat, resp. počtu promluv.

Počet aiml souborů	Doba zpracování	Počet uzlů grafu	Počet promluv	Velikost databáze
1	15,44s	36112	5032	2.70MB
10	29.87s	64348	8963	4.80MB
20	48.93s	96239	13612	7.30MB
30	66.34s	122043	18154	9.50MB
40	68.26s	128388	19284	10.00MB
50	94.67s	192846	27439	14.50MB
60	611.32s	712183	93376	51.00MB

Tabulka 6.1. Zpracování vstupních dat



Graf 6.1. Vliv počtu promluv na rychlost načítání dat

Statistika generování odpovědí

Dalším testovaným parametrem byla rychlost odpovědi (tj. jak rychle byl agent schopen najít reakci), její logická správnost a počet prohledaných uzlů. Pro porovnání naměřených výsledků je důležitý zejména počet uzlů ve vyhledávacím grafu, jelikož ten ovlivňuje rychlost hledání příslušné reakce. Testovaná množina promluv se rovnala vyhledávacímu grafu o velikosti 165 447 uzlů. Pro testování bylo vybráno $N = 8$ účastníků. Tito účastníci měli s robotem komunikovat v anglickém jazyce (viz. v kapitole 5 důvod použití volně dostupného korpusu), s použitím jednoduchých vět (oznamovacích i tázacích).

Uživatel	**Spokojenost uživatele (stupnice 1-5)**	**Maximální délka dialogu (počet logicky navazujících interakcí)**	**Procento logicky správných reakcí**	**Procento získaných odpovědí**	**Průměrná doba generování reakce [s]**	**Průměrný počet prohledaných uzlů**
Uživatel 1	1	5	86,00%	100%	1.72762	225
Uživatel 2	3	2	28,00%	94,00%	2.24581	282
Uživatel 3	2	3	32,00%	95,00%	3.01864	361
Uživatel 4	3	2	25,00%	96,00%	3.23346	432
Uživatel 5	3	3	38,00%	76,00%	6.02115	934
Uživatel 6	1	12	92,00%	100,00%	3.19588	326
Uživatel 7	4	2	22,00%	72,00%	12.03689	1467
Uživatel 8	1	4	64,00%	96,00%	3.43652	368

Tabulka 6.2. Statistika generování odpovědí

Transkript komunikace

Zde je uveden vybraný úryvek komunikace mezi uživatelem č. 6 (viz. tabulka 6.2) a chatovacím robotem, porovnání reakcí se vstupními soubory a zhodnocení reakcí.

Pořadí	**Promluva**	**AIML**
1	Uživatel**: Hi** (Ahoj) Robot**: hello** (Ahoj) **správně zodpovězeno**	<category> <pattern>HELLO</pattern> <template>**hello**</template> </category> <category> <pattern>**HI**</pattern> <template><srai>hello</srai></template> </category>
2	Uživatel: **How are you?** (Jak se máš) Robot**: Everything is going extremely well.** (Všechno jde skvěle)	<category> <pattern>**HOW ARE YOU**</pattern> <template> <random> - I am functioning within normal parameters. **Everything is going extremely well.** - Fair to partly cloudy.

	správně zodpovězeno	- My logic and cognitive functions are normal. - I'm doing fine thanks how are you? - Everything is running smoothly. </random> </template> </category>
3	Uživatel: **Nice to hear.** (To rád slyším) Robot**: Thank you for your kindness.** (Díky za tvou laskavost) **správně zodpovězeno**	<category> <pattern>**NICE ***</pattern> <template>**Thank you for your kindness.**</template> </category>
4	Uživatel: **What do you do?** (Co děláš?) Robot**: I talk to people on the web. What do you do?** (Mluvím s lidmi na webu. Co děláš ty?) **správně zodpovězeno**	<category> <pattern>**WHAT DO YOU DO**</pattern> <template> <get name="name"/>, **I talk to people on the web. What do you do?** </template> </category>
5	Uživatel: **I study.** (Studuji) Robot**: What subject are you studying?** (Jaký předmět studuješ?) **správně zodpovězeno**	<category> <pattern>**I STUDY**</pattern> <template> **What subject are you studying?** <think><set name="it"> <set name="does"><set name="topic">study</set></set></set> </think> </template> </category>
6	Uživatel: **I study math.** (Studuji matematiku) Robot**: Is math a hard subject?** (Je matematika těžký předmět?) **správně zodpovězeno**	<category> <pattern>**I STUDY ***</pattern> <template>**Is <star/> a hard subject?** <think> <set name="it"><set name="does"><set name="topic">studies </set></set></set></think> </template> </category>
7	Uživatel: **Yes, I do not like it.** (Ano, nemám ji rád) Robot**: Why not?** (Proč ne?) **správně zodpovězeno**	<category> <pattern>*** DO NOT ***</pattern> <template> <random> **Why not?** - Who does <star index="2"/>? </random> </template> </category>
8	Uživatel: **It is difficult.** (Je to složité) Robot**: How so?** (Jakto?) **správně zodpovězeno**	<category> <pattern>**IT IS DIFFICULT**</pattern> <template>**How so?**</template> </category>
9	Uživatel: **Teachers are hard.** (Učitelé jsou přísní) Robot**: All of them?** (Všichni?) **správně zodpovězeno**	<category> <pattern>*** ARE ***</pattern> <template> <random> - I think there might be a few exceptions. - All of them? </random><think> <set name="it"> <set name="topic"> <star/></set> </set></think> </template> </category>

Tabulka 6.3. Příklad vedené komunikace

7 Závěr

Cílem práce bylo seznámit se s metodami pro tvorbu síťových aplikací, dále pak s metodami generování neinformovaně vedených dialogů a tyto metody zhodnotit. Na základě těchto znalostí měly být vhodně zvolené technologie a postupy pro vytvoření víceuživatelského chatovacího robota a tohoto robota následně implementovat.

Cíle se podařilo splnit a výsledkem je chatovací robot, který podává očekávané výsledky v krátkém časovém intervalu. Při testech bylo zjištěno, že tyto výsledky jsou platné pro vyhledávací graf o přibližně 700 000 uzlech. Při větším objemu zdrojových dat se objevuje občasná nepřesnost získávaných odpovědí (v průměru 51%), což je záměrná funkcionalita, která zamezí chatovacímu robotu, aby hledal odpověď příliš dlouho.

Dalším krokem při vývoji této aplikaci by mohla být lokalizace chatovacího robota do českého jazyka. To by obnášelo vytvořit odpovídající bázi vstupních dat a úpravu aplikace (program by musel umět pracovat s diakritikou a musel by se vypořádat se skloňováním slov, např. slovo město má v češtině 11 tvarů). Dalším možným vylepšením by mohla být implementace úpravy vlastností robota a AIML souborů přímo na webovém rozhraní aplikace.

8 Přehled použitých zkratek

Zkratka	Celý název
XML	Extensible Markup Language
AIML	Artificial Intelligence Markup Language
JVM	Java Virtual Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTP	Hypertext Transfer Protocol
ACK	Acknowledgement
J2EE	Java platform, Enterprise Edition
JSP	Java Server Pages
URL	Uniform Resource Locator
ASP	Active Server Pages
PHP	Personal Home Page
AI	Artificial Intelligence
YAML	YAML Ain't Markup Language

Tabulka 9.1. Přehled použitých zkratek

Zdroje

[1] *A.L.I.C.E. AI Foundation*, **"AIML: Artificial Intelligence Markup Language".** Online: http://www.alicebot.org/aiml.html (available on May 17, 2012)

[2] *WALLACE, R.,* **"Official Alicebot AIML Wiki".** Online: http://alicebot.wikidot.com/ (available on May 18, 2012)

[3] *BUSH, N.,* **"Artificial Intelligence Markup Language (AIML)".** Online: http://www.alicebot.org/TR/2001/WD-aiml/ (available on May 18, 2012)

[4] *BUSH, N.,* **"Free AIML sets".** Online: http://aitools.org/Free_AIML_sets (available on May 12, 2012)

[5] *GOERZEN, J.*: **"Foundations of Python Network Programming"**, 2004

[6] *GILMORE, W.J.*: **"Beginning PHP 5 and MySQL: From Novice to Professional"**, 2004, ISBN-10: 1893115518

[7] *SKLAR, D. & TRACHTENBERG, A.*: **"PHP Cookbook"**, O'Reilly Media, 2003, ISBN-10: 1-56592-681-1

[8] *PUŽMANOVÁ, R.*: **"Moderní komunikační sítě od A do Z"**, COMPUTER PRESS, 1998, ISBN: 80-251-1278-0

[9] *BLUM, R.*: **"C# Network Programming"**, SYBEX Inc., 2003, ISBN: 0-7821-4176-5

[10] *REILLY, D. & REILLY, M.*: **"Java Network Programming and Distributed Computing"**, Addison-Wesley, 2002, ISBN: 0201710374

[11] *YOUNG, M.J.*: **"XML step by step"**, Microsoft Press, 2002, ISBN: 0735610207

[12] *BAEZA-YATES, R. & RIBERIO-NETO, B..*: **"Modern information retrieval",** Addison Wesley, 1999, ISBN-10: 020139829X

[13] *GREENBERG, J. & LAKELAND, J.R.*: **"a Methodology for Developing and Deploying Internet & Intranet Solutions",** Prentice Hall PTR, 1998, ISBN- 10: 0132096773

[14] **"Programovací jazyk Python".** Online: http://www.py.cz (available on Jun 12, 2012)

[15] *KOCOUREK, J.,* **"Výhody a nevýhody jazyka Python z pohledu programátora".** Online: http://www.itbiz.cz/python-pro-firmy (available on Jun 6, 2012)

[16] *TUTORIALSPOINT,* **"Python – network programming".**
Online: http://www.tutorialspoint.com/python/python_networking.ht
(available on Jun 6, 2012)

[17] *THE PHP GROUP,* **"PHP: Hypertext Preprocessor".**
Online: http://www.php.net/ (available on May 6, 2012)

[18] *PEREIRA, R.,* **"aerolito".** Online: https://github.com/renatopp/aerolito/
(available on May 6, 2012)

[19] *PETHERBRIDGE, N.,* **"RiveScript".** Online: http://www.rivescript.com/
(available on Jun 13, 2012)

[20] *WEIZENBAUM, J.*: **"ELIZA-A Computer Program for the Study of Natural Language Communication Between Man and Machine"**,
in: Communications of the ACM, pp. 36-45, 1966.