TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

DIPLOMOVÁ PRÁCE

2008 Radek Bartman

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T077 - Informační technologie

Systém hlasového dialogu s virtuální osobností

Voice dialogue system with virtual personality

Diplomová práce

Autor: **Bc. Radek Bartman**
Vedoucí práce: Prof. Ing. Jan Nouza, CSc.

V Liberci 1. 2. 2008

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Ústav informačních technologií a elektroniky Akademický rok: 2007/08

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jméno a příjmení: Radek Bartman

studijní program: M 2612 - Elektrotechnika a informatika

obor: 1802T007 - Informační technologie

Vedoucí ústavu Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu: **Systém hlasového dialogu s virtuální osobností**

Zásady pro vypracování:

1. Seznamte se se základy problematiky rozpoznávání a syntézy řeči, dále sémantické a gramatické analýzy textu a hlasově-textové interakce mezi člověkem a počítačem.
2. Vytvořte systém, který je schopen vést dialog mezi člověkem a virtuální osobností simulovanou počítačem. Dialog bude řízen otázkami člověka (respektive klíčovými slovy v těchto otázkách), na které bude počítač hledat odpovědi v databázi znalostí o dané osobnosti. Dialog by měl působit co nejpřirozeněji, tj. měl by pracovat s historií předcházejících fází dialogu, se schopností odpovědět i na nesrozumitelné dotazy a s možností převzít za určitých okolností iniciativu.
3. Systém vytvořte jako obecný. Zároveň připravte databázi znalostí, dotazů a odpovědí pro jednu až dvě vybrané osobnosti.

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum 16.5.2008

Podpis Bartman

PODĚKOVÁNÍ

Děkuji Prof. Ing. Janu Nouzovi, CSc. za odborné vedení diplomové práce a poskytnuté konzultace. Obzvlášť mu děkuji za vstřícný přístup a pomoc s přizpůsobením rozpoznávače řeči.

Děkuji Ing. Petru Červovi za pomoc s adaptací rozpoznávače řeči na můj hlas.

Děkuji také svým přátelům, rodičům a celé rodině za psychickou i finanční podporu při studiu.

Abstrakt

Předmětem této diplomové práce je navrhnutí a praktická realizace systému hlasového dialogu s virtuální osobností. Tento hlasový dialog by měl napodobovat normální rozmluvu mezi dvěma lidmi tak, aby umělá osobnost byla schopná reagovat na jakoukoliv větu, a měl by být schopen reagovat na dotazy v závislosti na historii zadaných dotazů. Systém reaguje vždy na nějakou otázku, iniciativu přebírá výjimečně. Systém umožňuje celou tvorbu dialogového schématu virtuální osobnosti bez jediného zásahu do zdrojového kódu programu, jelikož byl vytvořen jako platforma pro tvorbu libovolného množství virtuálních osobností s tím, že pro každou musíme dát dohromady databázi znalostí a reakcí na jednotlivé okruhy otázek, na které chceme, aby systém reagoval. Celá práce se skládá z několika programů, je k dispozici vizuální editor okruhů témat, program na skloňování slov a vlastní program určený pro komunikaci s člověkem. Tento program je možný provozovat také bez zapnutého rozpoznávání řeči a také bez syntézy řeči. Samotné rozpoznávání řeči a její syntézu tato diplomová práce neřeší, ale používá moduly zapůjčené z ústavu ITE.

Klíčová slova: dialog, vedení dialogu, rozpoznávání řeči, syntéza řeči.

Abstract

The aim of my thesis is the design and the practical realization of the voice system dialog with a virtual person. Mentioned dialog should simulate common conversation between two people in such a way that a virtual person is able to reply to any query. The voice system is supposed to be able to respond the query as a consequence of previous questions. The system does not initiate any conversation, i. e.: system puts the queries or starts the conversation only rarely.

The system enables the whole creation of the virtual person's dialog scheme without any intervention to its source code because the system was set up as a platform for creation of any arbitrary amount of virtual entities. It is inevitable to start off the database of knowledge and reactions for particular topics.

The whole thesis consists of various programmes, the virtual editor of particular topics, the programme for words declension and personal programme for human communication are also available. This programme is able to operate without speech recognition and also without any speech synthesis. The diploma thesis does not solve any problems with speech recognition and synthesis while the borrowed modules from ITE institute are used.

Keywords: dialogue, dialogue leading, speech recognition, speech synthesis.

Obsah

Seznam zkratek a termínů

C# název programovacího jazyku

MIT Massachusetts Institute of Technology

LINQ Language Integrated Query

PAC Phonetic Alphabet for Czech

IPA International Phonetic Alphabet

AIML Artificial Intelligence Markup Languague

XML Extensible Markup Language

DOM Document Object Model

SAX Simple API for XML

Úvod a motivace

Lidská řeč je nejstarším dorozumívacím prostředkem, který se osvědčil již mnoho tisíc let nazpět. Umožňuje velice rychlou komunikaci mezi dvěma lidmi nebo celou skupinou lidí. Vzhledem k tomu, že řeč je interaktivní, není problém pochopit i jiným způsobem složitě popsatelné věci, neboť v případě, že posluchač zjistí, že něčemu nerozumí, není problém požádat o upřesnění položením doplňující otázky. Řeč je jedním z nástrojů, kterými se vymaňujeme z živočišné říše a bezpochyby přispěla nad dominancí lidstva nad ostatními druhy. Postupem času se navíc řeč stala nejpoužívanějším dorozumívacím prostředkem a tím dodnes i přes rozmach moderních informačních technologií zůstává. Významnou výhodou lidské řeči je i to, že ji dokážeme používat bez přemýšlení a dělat během ní další činnosti, takže ji můžeme používat i v době, kdy jsme zaměstnáni jinou činností. Kromě vlastního obsahu poznáme z řeči i další věci, například příslušnost k určité sociální skupině (slang), aktuální náladu člověka, jeho zdravotní stav (chrapot), jeho emoce a také lze v hlase podle intonace vycítit nadsázku či despekt.

Není divu, že snaha ovládat počítač pouze za pomocí hlasu je zde již velice dlouho, neboť toto ovládání by zpřístupnilo práci s počítačem i lidem, kteří by měli s klasickým ovládáním veliké problémy. Hlasová komunikace s počítačem by umožnila pracovat s počítačem i těm lidem, kteří jsou negramotní a nedokážou se domluvit ani pomocí psaného textu, nebo lidem tělesně postiženým.

Přibližně od poloviny 90. let již jsou k dispozici systémy pro ovládání počítače hlasem, většinou se však jedná o systémy pouze reagující na klíčová slova. S rozvojem umělé inteligence se začínají realizovat také systémy, které umožňují plynulý dialog mezi počítačem a člověkem. Zatím tyto systémy nedokáží přesně pochopit význam jednotlivých slov, dost často však vystihnou podstatu věty a dokáží adekvátně zareagovat.

Možnosti těchto systémů nejsou pouze v ovládání počítače, ale je možné například naprogramovat virtuálního učitele, který má encyklopedické znalosti z mnoha vědních oborů a dokáže studentovi vysvětlit téměř libovolný pojem.

Tato diplomová práce se nesnaží o jakoukoliv úroveň hlasového ovládání počítače, pouze má za cíl vytvořit systém hlasového dialogu, který umožní co

nejpřirozenější komunikaci s počítačem s tím, že počítač bude vystupovat jako vybraná virtuální osobnost.

Cílem této práce je vytvořit systém hlasového dialogu s virtuální osobností, který bude schopen rozpoznávat mluvenou řeč od uživatele a reagovat na mluvené slovo, pokud možno tak, jako by reagoval normální člověk. Tento systém by měl být schopen zapamatovat si, co člověk řekl, a mít rozdílné reakce v závislosti na historii předchozího rozhovoru. Systém by měl mít připraveno také více možností odpovědí na jednu otázku, aby se příliš neopakoval. Měl by být také schopen reagovat na nesrozumitelné dotazy tak, aby působil co nejpřirozeněji. Systém by měl mít kromě klasického textového výstupu také výstup v podobě syntetické řeči a vstup v podobě mluveného slova tak, aby tento rozhovor byl co nejvíce autentický.

Systém by měl být navržený obecně tak, aby nebyl omezen pouze na jednu osobnost, ale aby bylo možné v něm vytvořit libovolné množství dialogových schémat pro různé osobnosti. Ke každé vytvořené osobnosti bude k dispozici databáze znalostí, na základě kterých se bude rozhodovat a na základě kterých bude reagovat. Systém by měl primárně reagovat na zadané dotazy, pouze za určitých okolností by měl převzít iniciativu.

Vlastní rozpoznávání řeči a její syntézu tato diplomová práce řešit nemá, příslušné softwarové moduly budou zapůjčeny ústavem Informačních technologií a elektroniky.

Jednotlivé virtuální osobnosti budou tvořeny konfiguračními XML soubory s definovanou strukturou. XML soubory mají tu výhodu, že se dají v případě nutnosti editovat v běžném textovém editoru.

Vlastní program bude vytvořen v inovovaném programovacím jazyce C# 3.0, který přináší prvky funkcionálního programování a obsahuje dotazovací jazyk LINQ, který bude jistě s výhodou použit při vyhledávání vhodných odpovědí na otázky. Jako vývojové prostředí bude využito Visual Studio 2008.

1 Existující virtuální osobnosti

Systémy, které se snažily napodobovat komunikaci s virtuálním člověkem, jsou již velmi staré a odjakživa se těšili velkému zájmu programátorů i uživatelů. Již v roce 1966 vytvořil německý matematik Joseph Weizenbaum první verzi svého programu Eliza (viz [1]). Tehdy se pro komunikaci s virtuální osobností vžil název *chatbot*. Tento program napodoboval rozhovor pacienta s psychiatrem a byl dokonce tak reálný, že sekretářky a další personál na MIT (Massachusetts Institute of Technology), trávili čas tím, že několik hodin komunikovali s tímto „terapeutem", aniž by poznali, že se nejedná o člověka ale o počítač, a svěřovali mu své osobní problémy. Mysleli si, že program jim skutečně rozumí, ačkoliv program využíval pouze několika jednoduchých triků a i jeho tvůrce byl překvapen tím, jak jednoduchý program dovede přinutit lidi k prozrazení soukromých údajů. Program využíval například triku, kdy pokud uživatel napíše „Potřeboval bych X" tak program mu na to odpoví „Čemu by pomohlo, kdybys měl X". Takovéto větné konstrukce dávájí smysl vždy, X mohou být například peníze, přátelé, láska, jídlo či čas. Program měl k dispozici i několik klíčových slov, která když rozpoznal, tak měl připravenou vždy nějakou odpověď. Například pokud nalezl slovo „matka", tak odpověděl otázkou „řekněte mi více o své rodině". Pokud program nenarazil na žádnou známou větnou konstrukci, tak se vrátil automaticky k předchozímu tématu.

Následníků Elizy se objevilo nespočet:

- V roce 1972 to byl PARRY(viz [2]), virtuální osobnost, která simulovala paranoidního schizofrenika.
- V roce 1990 se objevil Dr. Sbaitso od Creative Labs (tento program, koncipovaný jako hra, měl dokonce zvukový výstup)
- Nejdokonalejší virtuální osobností dneška je nejspíš ALICE(viz [3]). Tento program několikrát zvítězil v Turingově testu.[1]

Dalších programů je nespočet, zmiňme alespoň InteliWISE AVATAR (viz [4]), jenž vyniká především grafickým zpracováním virtuální osobnosti a také kvalitní syntézou

[1] Turingův test se snaží dát odpověď na otázky: „Mohou počítače myslet? A pokud je počítač schopen myslet, lze to rozpoznat?" Jako odpověď na tento postup vyvinul Turing jednoduchý postup. Dnes používaný upravený Turingův test funguje tak, že rozhodčí pokládá různé otázky dvěma systémům a snaží se určit ze kterého odpovídá počítač a ze kterého odpovídá člověk. Pokud rozhodčí nedokáže správně určit kde je člověk a kde program, je program úspěšný.

řeči, a program UltraHAL (viz [5]), jenž funguje jako osobní asistentka. Tento program má na rozdíl od většiny jiných implementováno jak hlasovou syntézu, tak rozpoznávání řeči, takže se s ním dá komunikovat podobně jako s normální sekretářkou.

Českých a slovenských projektů na tomto poli bylo také mnoho, jmenujme alespoň Pokyd [6], Ludvík [7] nebo WinKec [8].

Zmíněný program ALICE je naprogramovaný pomocí jazyka AIML (Artificial Intelligence Markup Languague), jenž vznikl odvozením z jazyka XML. AIML je značkovací jazyk podobně jako HTML, ale na rozdíl od něj klade důraz na rychlé vyhledávání v datech a propojeni s ostatními značkovacímy jazyky (např. XHTML). AIML obsahuje 3 základní tagy:

- <category> - kategorie, jednotka znalostí programu
- <pattern> - vzorek, slovo či celá věta, který program najde v napsané větě
- <template> - připravená odpověď na zadaný vzorek

Původní program ELIZA byl vybaven pouze okolo 200 kategoriemi, program ALICE jich obsahuje přes 40000, takže dokáže reagovat na velké množství různorodých otázek.

Programů, které by dokázali odpovídat na mluvenou řeč, je však velice málo, jmenujme alespoň program RoboMatic X1[9], zmíněný program UltraHAL či Projekt Švejk[10] z roku 2002, který byl naprogramován na Technické univerzitě v Liberci.

2 Teoretické základy a použité prostředky

2.1 Možné přístupy rozpoznávání psaného textu a způsoby reakce na něj

Psaný text je možné rozpoznávat mnoha způsoby. Úplně nejjednodušší by byl způsob, kde by člověk neměl vůbec žádnou možnost interakce s počítačem. Pouze by si na začátku vybral téma, na které se bude bavit, a počítač by mu pokládal předem připravené otázky. Podobné systémy se v minulosti již objevili, například program RACTER(viz [11]) byl podle tvrzení autora schopen napsat knihu sám o sobě.

Lepší je určitě přístup kdy se rozeznávají konkrétní klíčová slova a poté se podle předem připravených pravidel „řekne" nějaká odpověď. Toto již lze používat, problémem však je, že počítač nedokáže upravit svou odpověď přesně podle toho, co člověk řekl, takže takovýto program dokáže předstírat člověka jen malou chvíli.

První virtuální osobnost Eliza používala ještě rafinovanější způsob rozpoznávání textu. Stačilo ji, když rozpoznala sekvenci několika slov zpravidla ze začátku věty a pokud za nimi následovala jakákoliv další sekvence slov, tak ji šikovně využila. Tento postup lze použít na překvapivě velkém množství vět tak, aby to dávalo smysl. Např. na větu „Já jsem velice unavený" lze snadno připravit odpověď ve tvaru „Opravdu si myslíš, že jsi velice unavený".

Tento způsob se ukázal do dnešní doby v podstatě jako nepřekonaný, pouze prodělal pár evolučních změn. V prvé řadě přibylo daleko více konstrukcí, na které chatbot dokázal odpovědět, a také se přidala možnost použít předem připravené proměnné, takže chatbot dokáže mít rozdílné reakce v závislosti na těchto proměnných. Dnes již je u spousty programů běžné i to, že dokáží tyto proměnné aktualizovat podle hovoru. Kromě jména si například dokážou zapamatovat i v kolik hodin má člověk naplánovanou nějakou akci (viz zmíněný chatbot UltraHAL).

Všechny zmíněné přístupy k těmto chatbotům se mohou na první pohled pro neznalého uživatele jevit jako skutečná osobnost, ve skutečnosti však počítač nikdy neví, na co se ho člověk ptal, a dokonce ani neví, co odpovídá. Tento fenomén se nazývá paradox čínské místnosti a v 80. letech ho prokázal filosof John Searle. Ten umístil do uzavřené místnosti jednoho člověka a nechal mu tam posílat proužky papíru, na kterých byly napsány čínské znaky. Tento člověk pak měl tyto čínské znaky přepsat

na tabuli v angličtině. Vtip byl v tom, že tento člověk čínsky vůbec neuměl, pouze byl vybaven manuálem, kde měl napsáno, že pokud uvidí konkrétní znak, tak ať napíše nějaký konkrétní text na tabuli. Všichni lidé, co ho pozorovali v oddělené místnosti, ho však měli za znalce čínštiny, ačkoliv tento člověk neuměl vůbec ponětí, o čem je obsah komunikace.

Bohužel zatím neexistuje systém, který by přesně věděl a uvědomoval by si, o čem se zrovna komunikuje. Naštěstí to na druhé straně člověk není schopen nijak snadno rozeznat. Objevují se však již nyní systémy, které se snaží alespoň pochopit větnou stavbu a jejich reakce jsou pak přesnější. Zpravidla se snaží hlavně určit podmět a přísudek ve větě a reagovat pak podle něj. Tyto ambice, ačkoli nenaplněné, měl například český IQ Pokyd.

2.2 Rozpoznávání řeči

Rozpoznávání řeči a její porozumění na počítači je velice netriviální úloha. Když přichází řeč v podobě zvukových pulzů do lidského sluchového ústrojí, máme k dispozici poměrně dokonalý aparát pro její porozumění. Pro potřeby počítačového zpracování si ho můžeme dle [12] rozdělit na několik úrovní:

1) Akustická analýza – je to nejnižší úroveň, na této úrovni provádíme detekci vlastní řeči, její oddělení od ostatních zvuků.
2) Fonetická analýza – na této úrovni se zachycený signál registruje jako sekvence hlásek či slabik. Hlásky ani slabiky však ještě nenesou žádný význam.
3) Lexikální analýza – na této úrovni se ke zvukům odpovídajícím jednotlivým hláskám a slabikám přiřazují slova, která již nesou konkrétní význam. K tomu abychom mohli tato slova dát dohromady, potřebujeme znát konkrétní slovník daného jazyka.
4) Syntaktická analýza bere ohled na vzájemné vztahy mezi jednotlivými slovy a umožňuje nám preferovat jen takové kombinace slov, které jsou v daném jazyce přípustné a které mohou tvořit významový celek. Tímto významovým celkem je věta.
5) Sémantická analýza – skutečný význam věty nám zjišťuje sémantická analýza. Ta zajišťuje, že věty jako „Je půl šesté." a „Je pět hodin třicet minut." budou interpretovány jako významově shodné.

6) Pragmatická analýza – je na nejvyšší úrovni tohoto rozdělení a bere v úvahu také kontext, ve kterém byla věta řečena, a postoj řečníka. Např. věta „To jsi mi udělal radost." Myšleno vážně nebo ironicky.

Toto rozdělení je samozřejmě pouze nedokonalým modelem, zatím se ještě nepodařilo přesně zjistit, jakým způsobem přesně člověk vnímá a analyzuje text. Zejména se ukazuje, že se těchto 6 úrovní nevyhodnocuje postupně, ale částečně paralelně (viz [13]).

Výhodou člověka oproti počítači v rozpoznávání řeči je, že člověk je vybaven vlastní inteligencí, motivací, různými vědomostmi a znalostmi kontextu, takže mu nečiní problém, když určitému úseku řeči nerozumí, je schopen domyslet si tuto část. Producentem řeči je navíc výhradně člověk, a tak je řečový signál značně nedeterministický, variabilní v závislosti na konkrétní osobě, jejím momentálním fyzickém a psychickém stavu, na jejím sociálním původu a vyjadřovacích schopnostech.

Počítač je vždy vybaven jen takovým korpusem znalostí, které mu dal do vínku jeho programátor. I kdyby programátor dal počítači do vínku všechny své znalosti, tak stále bude počítač bez emocí, bez znalostí kontextu, bez inteligence a bez vlastního postoje, což ho oproti člověku znevýhodňuje. Místo porozumění konkrétního sdělení počítač pouze dekóduje jednotlivé zvuky a převádí je do textové podoby.

Mluvená řeč je také na rozdíl od své psané podoby spojitá bez jasně definovaných hranic. Hranice neexistují mezi jednotlivými slovy ani mezi hláskami.

Elementárním prvkem pro rozpoznávání spojité řeči jsou většinou tzv. fonémy – foném je nejmenší jednotka řeči, která nám může rozlišovat jednotlivá slova. Pomocí fonémů můžeme zapsat zvukovou podobu jakéhokoliv slova. Fonémů je v češtině dle abecedy PAC (Phonetic Alphabet for Czech – podrobnosti viz [12]) 41, z toho 40 hlásek a pauza. Výhodou PAC je její snadná čitelnost, kdy většina psaného textu má přepis v abecedě PAC velice původní originálu, takže ji mohou po krátkém zaškolení na rozdíl od mezinárodní fonetické abecedy IPA (International Phonetic Alphabet, mezinárodní fonetická abeceda) používat i neodborníci.

Modely fonémů je třeba natrénovat na záznamech řeči obsahující tisíce jednotlivých realizací každé hlásky, jelikož jejich zvuková realizace značně závisí na kontextu v rámci slova (na okolních hláskách).

Pro účely rozpoznávání spojité řeči je potřeba mít také slovník slov, která se mohou v dané řeči vyskytovat (jelikož v řeči neexistuje hranice mezi slovy, takže nejde udělat pouhý přepis fonémů do textové podoby), a také fonetický přepis všech jeho slov. To je obecně u češtiny problém, jelikož díky ohebnosti slov má čeština přes 500 000 různých tvarů slov.

Dalším krokem je vytvoření jazykového modelu, například bigramového, který se na velkém množství textu naučí, jaká je pravděpodobnost, že slovo A bezprostředně předchází slovu B pro všechna slova ze slovníku. Tento jazykový model je dobré natrénovat na velice objemném souboru vět, jelikož kombinací jednotlivých dvojic slov je velice mnoho a je vhodné, aby bylo zastoupeno co největší množství těchto kombinací. V tomto modelu je samozřejmě mnoho kombinací, které nebyly nikdy viděny, proto se musí provést jeho vyhlazení. Podrobněji např. viz. [14].

Zvukový signál již můžeme nyní rozpoznávat. Používaný je zejména Viterbiho dekodér. Tento dekodér nám určuje nejpravděpodobnější posloupnost slov. Pro malé slovníky umožňuje spočítat pravděpodobnosti všech kombinací slov; pro obsáhlejší slovníky je však nutné průběžně zahazovat posloupnosti slov, které již mají velice malou pravděpodobnost, jelikož kombinací by existovalo příliš velké množství a nebylo by možné v rozumném čase všechny prověřit.

Viterbiho dekodér nám již dá odpověď na otázku, která posloupnost slov nejvíce odpovídá původnímu zvukovému signálu. Problematika rozpoznávání řeči je velice obsáhlé téma a zde je uveden pouze jakýsi nástin, více informací je k dispozici např. v [12] a [15].

2.3 Syntéza řeči

Syntéza řeči počítačem představuje úlohu převedení řeči (původně textově zapsanou) do zvukové podoby. Oproti rozpoznávání řeči má tu výhodu, že i když řeč produkuje nedokonalý počítač, tak posluchačem je inteligentní člověk, takže většinou nebývá problém syntetizované řeči porozumět ani u velmi starých a jednoduchých systémů.

Hlasový trakt člověka je tvořen soustavou několika dutin (dutina hrdelní, nosní a ústní) a také do něj zahrnujeme i koartikulační orgány – jazyk, zuby, rty a hlasivky. Jeho vlastnosti jsou pak dány fyziologickými vlastnostmi těchto orgánů, a to jejich délkou, tvarem a průřezem. Parametry těchto orgánů jsou pro každého člověka

jedinečné a navíc se často mění i při vyslovování různých fonémů. Tyto orgány mají své rezonační frekvence, které se projevují v modulové frekvenční charakteristice jako maxima, říkáme jim *formanty*. Minimům ve frekvenční charakteristice na antirezonančních frekvencích odpovídají antiformanty.

Z hlediska historie syntézy řeči se objevilo několik modelů:

- Formantový syntezátor (Formant je spektrální složka řečového signálu, kterou lze simulovat pásmovou propustí. Řazením těchto filtrů za sebou se generuje hláska.)
- Syntezátor s tabelovanými zvuky – skutečná řeč se navzorkuje a A/D převodníkem převede na malé úseky řeči. Tyto úseky řeči se pak řadí za sebe a vzniká tak syntetická řeč.
- Lineárně prediktivní kódování – hlasový trakt je modelován pomocí přenosové funkce obsahující pouze póly.
- Metoda pitch-synchronního spojování řečových úseků.
- Sinusové kodéry signálu.
- Kepstrální syntezátor.

V poslední době se také díky snížení ceny paměťových medií používá také metoda přímé produkce nahrané řeči. Nahraje se řádově alespoň několik desítek hodin řeči, určí se k ní hranice jednotlivých slov a textový přepis a poté, když chceme reprodukovat nějaký text, tak se snaží vyhledat co nejdelší úsek, který má zaznamenaný a tento úsek přehraje. Pokud nějaké slovo nemá zatím nahrané, použije se některý z parametrických syntezátorů vyjmenovaný výše.

Ačkoliv syntéza řeči prodělala v posledních letech veliký vývoj, kvality lidského protějšku zatím nedosahuje, a to zejména proto, že počítač reprodukuje pouze text ve zvukové podobě bez znalosti jeho obsahu a bez zaujmutí vlastního stanoviska k němu. Zejména v prozodii (melodii textu) jsou ještě značné rezervy. Více informací je k dispozici v [12] a [15].

2.4 XML

XML (eXtensible Markup Language, česky rozšiřitelný značkovací jazyk) je velmi oblíbený formát pro publikování a výměnu dokumentů mezi aplikacemi. XML je

jazyk pro popis dat, jeho výhodou je, že je samopopisující a na rozdíl od HTML umožňuje důsledné oddělení samotné datové informace od její grafické reprezentace.

Přestože dnes již mnohé dokumenty obsahují obrázky, hudbu či video, mnohé dokumenty jsou stále založeny hlavně na textu. XML je formát, který umožňuje snadno takovéto informace popisovat. Např. i nový formát v MS Office 2007 OOXML využívá výhod XML formátu. XML není majetkem žádné soukromé firmy, ale byl vyvinut konsorciem W3C a navrhnut na základě zkušenosti s předchozími značkovacími jazyky.

XML je ideální formát pro ukládání strukturovaného a semi-strukturovaného textu. Dokument XML obsahuje specifické instrukce nazývané tagy, elementy a entity, které označují jednotlivé části dokumentu. Zde uvedený úryvek XML souboru je převzatý z vlastní aplikace:

```
<Zaznam Id="46" KolikratPouzito="5">
      <Text>Já se mám dobře.</Text>
      <ZdrojoveTema>mitse</ZdrojoveTema>
</Zaznam>
```

Barevně je zde zvýrazněné značkování XML. XML soubory musí dodržovat pevnou strukturu, takže například ke každému počátečnímu *tagu elementu* (zde například <**Zaznam**>, <**Text**> a <**ZdrojoveTema**>) musí existovat koncový *tag*. Počáteční *tag* a koncový *tag* spolu s daty mezi nimy tvoří jeden *element*. Tímto způsobem XML identifikuje jednotlivé objekty. Jednotlivé elementy mohou obsahovat libovolný počet vnořených elementů, a tak se dá vytvořit stromová struktura dat. Celý dokument musí být vložen do jediného *elementu dokument*. Každý *element* ale musí být plně vnořen do jiného *elementu*, není možné, aby *element* měl počáteční *tag* v jednom *elementu* a koncový v jiném.

Každý *element* může navíc obsahovat ještě další doplňující informace o svém obsahu, tzv. *atributy*. Těchto informací (*atributů*) může mít každý *element* více, proto musí mít každý *atribut* přiřazeno své jméno. Ve výše uvedeném příkladu tak například *element* **Zaznam** má *atributy Id* (jednoznačně identifikuje celý *element*) a *KolikratPouzito* (tento *atribut* nám říká, kolikrát byl daný **Zaznam** použit v programu). *Atribut Id* má hodnotu „46" a *atribut KolikratPouzito* hodnotu „4".

XML soubory mají spoustu dalších vlastností a možností, jako jsou komentáře, deklarace a zpracovací instrukce. Informace o těchto prvcích lze nalézt například v [16].

Mnoho vývojářů, kteří chtěli používat XML soubory v době, kdy nebyly tolik rozšířené, byli nuceni napsat si vlastní API pro práci s XML dokumenty. Jelikož spousta vývojových skupin vyvinula pro tento účel mnoho odlišných API, vznikl problém se vzájemnou kompatibilitou. Naštěstí komunita kolem XML tento problém relativně včas zachytila a definovala několik standardů.

2.4.1 XML SAX

SAX (Simple API for XML) umožňuje sériový přístup ke XML. Je to model, který je řízen událostmi. Parser sekvenčně prochází XML soubor, a pokud narazí na jakoukoliv významnou položku, jako jsou počáteční tagy elementů, komentáře, atributy, atd., vyvolá příslušnou událost. Programátor pak musí definovat třídu, která implementuje rozhraní *DocumentHandler,* a předat na ní odkaz parseru. V tomto rozhraní jsou pak deklarovány metody, které se volají v případě různých událostí.

Ačkoliv tento model vypadá složitě, jeho výhody vyniknou v případě zpracování zejména složitějších XML souborů. Oproti modelu DOM (viz následující kapitola) vyniká také rychlostí zpracování XML souboru.

2.4.2 XML DOM

DOM (Document Object Model) je objektově orientovaná reprezentace XML nebo HTML dokumentu. Vznikl z důvodu nutnosti standardizovat přístup webových prohlížečů k jednotlivým elementům html stránky.

DOM umožňuje přístup k dokumentu jako ke stromu, což je zároveň datová struktura obvykle používaná XML parsery. Tato technologie vyžaduje nahrání celého dokumentu do paměti, z čehož plyne, že její optimální využití je tam, kde je potřeba k jednotlivým částem dokumentu přistupovat v náhodném pořadí nebo opakovaně.

Rozhraní DOM v zásadě definuje rozhraní používaná pro správu uzlů (*nodes*). Každý element, atribut, komentář, entita i celý dokument jsou reprezentovány uzlem (*node*). Rozhraní *Node* definuje řadu metod, které by se daly rozdělit do třech širokých kategorií podle toho, co zjišťují. Zaprvé každý uzel má svou charakteristiku, jako je jméno, typ a hodnota. Potom každý uzel má také pozici v kontextu stromu a poskytuje přístup k příbuzným uzlům. Nakonec má také každý uzel schopnost modifikovat svůj obsah – uzly reprezentující jeho děti.

Přístup ke XML souborům pomoci DOM byl využit v aplikaci XML Builder, viz kapitola 4.2.

2.5 C# 3.0 a jeho specifika

Jako programovací jazyk byl vybrán jazyk C#, konkrétně ve verzi 3.0. V této verzi přišel Microsoft s mnoha inovacemi a tento programovací jazyk se poněkud vymezil oproti ostatním klasickým programovacím jazykům. Jedná se zejména o:

- Rozšiřující metody (Extension methods). Jsou to metody, kterými lze rozšířit stavající datové typy (objekty) o nové metody. Metody mohou rozšiřovat i objekty, které využívají vybrané rozhraní.
- Automatické proměnné. Umožňují se vyhnout klasické definici metod get a set, překladač si vnitřek těchto metod již vygeneruje sám. Příklad: *public string Name { get; private set; }*
- Lambda výrazy. Lambda výrazy vznikly, aby nahradili složitý zápis delegátů. Místo *seznam.Where(delegate(Foo x) { return x.Size > 10; });* tak můžeme psát *seznam.Where(x => x.Size > 10);*
- Nové klíčové slovo *var* můžeme používat tam, kde si nejsme jisti s tím, jakého typu je daná proměnná. Typová bezpečnost je ale stále zachována, jelikož překladač si správný typ určí ještě během překladu.
- Language Integrated Query (LINQ) neboli jazyk pro integrováné dotazování. Jedná se o největší přínos jazyka C# ve verzi 3.0, neboť pomocí jazyka LINQ se velice zjednodušuje práce s téměř jakýmikoliv daty.

2.5.1 Jazyk LINQ

Jazyk LINQ se v mnohém podobá jazyku SQL, využívá obdobných konstrukcí a jeho účel je také podobný, slouží pro manipulaci s daty, jejich třídění, spojování a rychlému vyhledávání. Jednoduchá aplikace demonstrující práci pomocí Linq může vypadat například takto:

```
string[] slova = { "Ahoj", "Čau", "Dobrý den", "Nashledanou",
                   "Dobrou noc" };
var kratkaSlova = from c in slova
                  where c.Length < 5
                  orderby c.Length
                  select c;
foreach (string slovo in kratkaSlova)
{
      Console.WriteLine(slovo);
}
```

V tomto ukázkovém kódu se nejprve do pole *slova* přiřadí na prvním řádku několik různě dlouhých řetězců, poté se na třetím řádku pomocí klíčových slov *from* a *in* přikáže, aby počítač postupně prošel celé pole a každou hodnotu z tohoto pole dočasně přiřadil do proměnné *c* (typu *string*). Poté se pro každý řetězec *c* zjišťuje, jestli je jeho délka menší než 5, a pokud ano, tak se vybere. Všechny vybrané řetězce se ještě setřídí podle délky a poté se vytisknou na obrazovku. Za klíčovým slovem *var* a konstrukcí *var kratkaSlova* se ve skutečnosti skrývá *IOrderedEnumerable<string> kratkeSlova*, a to již v době překladu. Jak je vidět tak koncept jazyka LINQ nám umožňuje manipulovat s daty podobně jako v přirozené řeči.

LINQ má v současné volbě dvě metody zápisu, zde předvedená metoda je sice přehlednější, ale více „upovídaná" než metoda druhá:

```
string[] slova = {"Ahoj", "Čau", "Dobrý den", "Nashledanou",
                  "Dobrou noc" };
var kratkaSlova = slova.Where(c => c.Length < 5).
                        OrderBy(c => c.Length);
foreach (string slovo in kratkaSlova)
{
      Console.WriteLine(slovo);
}
```

V této verzi se s výhodou používá rozšiřujících metod a zmíněných lambda výrazů. Tímto způsobem se lze dotazovat přesněji, jelikož jeho vyjadřovací možnosti jsou bohatší. LINQ je má již v základu definovaných velké množství metod rozšiřujících typ *IEnumarable<T>*:

- *Select* – slouží k výběru hodnoty, kterou chceme použít.
- *Join* – slouží ke spojení více poskytovatelů dat.
- *GroupBy* – slouží pro rozdělení dat do více skupin podle určitého klíče.
- *Where* – slouží k omezení výběru provků podle specifikované podmínky.
- *OrderBy*, *OrderByDescending* – slouží ke specifikaci třídění, umožňují výběr elementu, podle kterého se má třídit.
- *First*, *Last* – slouží k výběru prvního nebo posledního prvku z kolekce.
- *ElementAt* – slouží k výběru prvku podle udaného indexu.
- *Count* – vrací počet prvků v kolekci.

- *Union*, *Intersect*, *Except* - jsou definovány jako množinové operace sjednocení, rozdíl a průnik.
- *Sum*, *Min*, *Max*, *Average* – vrací součet, minimální, maximální či průměrnou hodnotu z dané kolekce.

Uvedli jsme si výčet těch nejdůležitějších rozšiřujících metod a je vidět, že záběr LINQu je skutečně veliký. Velikou výhodou je, že LINQ je možné používat prakticky nad všemi daty. Již od základu je dodáván LINQ se třemi poskytovateli:

- LINQ to Objects – slouží pro dotazování nad daty, která již jsou v paměti, tj. nad poli a dalšími třídami implementující rozhraní *IEnumerable<T>*. Dotazovací engine je spuštěn spolu s programem a umožňuje lokální dotazování. Tento způsob dotazování není dynamický, takže jakmile se jeden dotaz vyhodnotí a vrátí výslednou množinu, tak se do něj již nepromítají změny v původních datech.
- LINQ to SQL – umožňuje dotazování nad databázemi využívající jazyk MS SQL. Jelikož tyto databáze mají svůj vlastní dotazovací jazyk SQL, není zde přímo nasazen dotazovací engine LINQ, ale místo toho se příkazy LINQu mapují na odpovídající příkazy SQL. Jelikož jsou ale data v těchto databázích uložena jako relační, musí být ještě nasazen tzv. mapper těchto dat na objektová data, která používá LINQ. Výhodou použití LINQu místo klasického přístupu je zejména objektový pohled na data.
- LINQ to XML – slouží pro práci s daty uložených v XML souborech, nepoužívá ani programování založené na DOM nebo SAX, ale jde novou cestou, kdy je k datům přistupováno plně objektově. Jelikož práce s XML soubory pomocí tohoto přístupu využívá značná část výsledné aplikace, věnujeme tomuto přístupu samotnou podkapitolu.

Existují i další poskytovatelé dat, např. PHPLinq, PLinq (projekt určený pro výpočet s využitím několik vláken), LINQ to Amazon a další. Více informací o dalších projektech viz například [17].

2.5.2 LINQ to XML

Ačkoliv mohl Microsoft napasovat LINQ na již existující modely práce s XML soubory, neudělal tak a navrhnul zcela nový koncept práce s nimi. Práce s XML

soubory pod LINQem je opravdu velice snadná a připomíná funkcionální programovací jazyky. Ukázka vytvoření jednoduchého XML souboru:

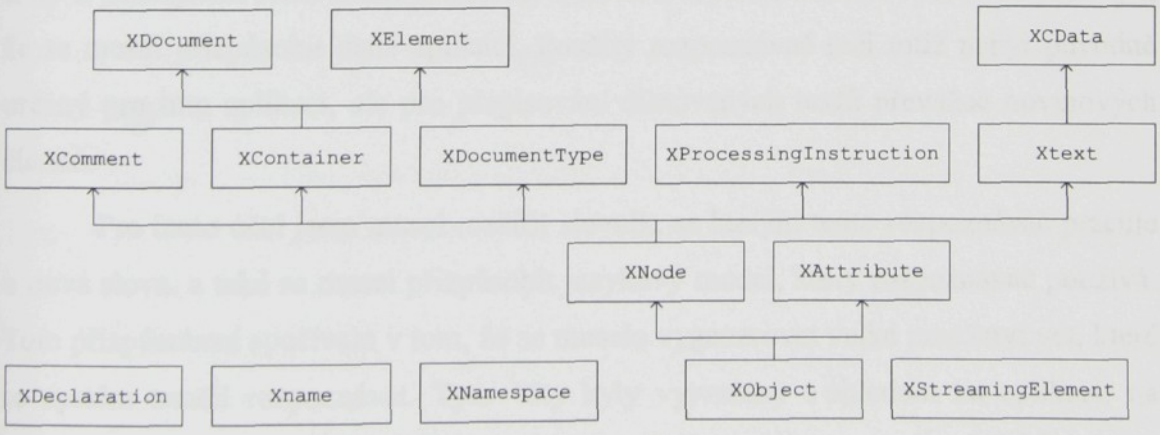
```
XElement xUzivatel =
      new XElement("Uzivatel",
            new XAttribute("Zamestnani", "Voják"),
            new XElement("Jmeno", "Otto"),
            new XElement("Prijmeni", " Katz "));
```

Tento krátký úsek kódu vygeneruje element XML, jehož obsah vypadá takto:

```
<Uzivatel Zamestnani="Voják">
      <Jmeno> Otto </Jmeno>
      <Prijmeni>Katz</Prijmeni>
</Uzivatel>
```

S API DOM něco takového nebylo možné, pokud by člověk chtěl vytvořit pouze XML element bez mateřského XML dokumentu, neměl žádnou možnost. Bylo to celkem omezující, protože pokud člověk chtěl vytvořit opravdu právě jen jeden element, DOM API ho nutilo k práci, která byla zbytečná. S Microsoft XML API můžete vytvářet Elementy, Atributy, komentáře i všechny ostatní prvky XML bez mateřského elementu.

Element reprezentuje objekt XElement, atribut má svůj obraz v XAttribute a celý dokument reprezentuje objekt XDocument. Hierarchii jednotlivých objektů ukazuje následující schéma:



Obrázek 1: Objektová struktura XML API od Microsoftu

Na tomto schématu jsou rodiče zobrazeni úplně dole a jejich potomci jsou zřejmí použitím šipek.

Ukázka jednoduché práce s XML pomocí Microsoft XML API z vlastní aplikace:

```
var vsechnyVety =
xProjekt.Element("Projekt").Element("SeznamVet").Elements("Zaznam");

var vety = vsechnyVety.Elements("ZdrojoveTema").Where(c =>
{
      foreach (String tema in aktualniTemata)
      {
      if ((String)c.Value == tema)
      return true;
      }
      return false;
}
).Select(c => c.Parent).ToArray();
```

Zde se v první části vyberou všechny věty ze souboru s tématy (viz například soubor *temata.xml* z vlastní aplikace) a posléze se z těchto všech vybraných vět vyberou pouze ty, které mají za zdrojové téma jedno z témat uložených v seznamu *aktualniTemata*.

2.6 Rozpoznávač řeči a syntezátor řeči

Jak již bylo řečeno, tato práce neřeší vlastní rozpoznávání ani syntézu řeči, ale používá moduly, které za tímto účelem zapůjčil Ústav informačních technologií a elektroniky. Zatímco napojení na syntezátor řeči spočívalo pouze v tom napsat obalové třídy k existujícím DLL knihovnám, napojení na rozpoznávač řeči bylo o to složitější, že se musel přizpůsobit dané aplikaci. Použitý rozpoznávač řeči totiž nebyl původně určený pro tuto aplikaci, ale pro přepisování diktovaných textů převážně novinových článků.

Pro tento účel jsem musel rozšířit slovník, se kterým tento rozpoznávač pracuje o nová slova, a také se musel přizpůsobit jazykový model, který rozpoznávač používá. Toto přizpůsobení spočívalo v tom, že se muselo vygenerovat velké množství vět, které se systém naučil rozpoznávat. Tyto věty byly vytvořeny s ohledem na aplikaci, na kterou byl tento systém určen, takže se jednalo zejména o různé konverzační fráze.

Pro vylepšení úspěšnosti rozpoznávání byl další krok také adaptace systému na můj hlas. Pro účely adaptace jsem musel namluvit několik desítek vět a z těchto zvukových záznamů se poté vytvořil nový hlasový model, který aplikace používá.

3 Vedení dialogu

Mechanismus vedení dialogu použitý v této práci musel zohlednit, že mu vstupem nejsou kompletně gramaticky a morfologicky správné věty, ale věty, kde se procento chyb ukazuje jako nezanedbatelné. Díky tomu se složitost této úlohy poněkud zvětšila, jelikož je poté výrazně těžší připravit správnou odpověď. V programu se poté musí uvažovat i v běžné řeči nereálné varianty vstupního textu. Použitý rozpoznávač řeči totiž nebyl původně připraven pro takovýto typ nasazení, ale původně byl navrhnut jako rozpoznávací systém určený pro diktování textů do počítače (viz předchozí kapitola).

3.1 Rozdělení na tematické celky

Základním kamenem mého řešení dialogového systému jsou tematické celky, ve kterých se rozhoduje o směrování rozhovoru. Tyto tematické celky označuji v aplikaci jako „*Tema*". Téma může být použito buď jako zdrojové, nebo jako cílové (bude vysvětleno dále). Každý projekt má definováno jedno dominantní téma, které se použije již při startu programu. Toto téma se musí jmenovat „hlavni". Program si uschovává seznam aktuálních témat ve své interní kolekci a podle nich se rozhoduje. V systému se rozeznávají dva druhy vět. Jedná se o větu ve smyslu otázky, tj. o větu, na kterou má systém zareagovat (pro označení tohoto druhu věty budu dále v textu používat označení buď věta či otázka) a o větu, která má být reakcí systému na danou otázku (tento typ věty budu dále v textu označovat jako odpověď). Otázky jsou všechny uloženy v souboru s projektem v elementu *SeznamVet* a odpovědi v elementu *SeznamOdpovedi*.

Abychom si lépe ilustrovali základní rozhodující mechanismus, bude vhodné nejprve si ukázat, jak jsou věty a odpovědi přesně uloženy v systému.

```
<SeznamVet>
      <Zaznam Id="13" KolikratPouzito="0" Priorita="0">
            <Text>(ahoj)|(čau)|(nazdar)|(zdar)|(čágo)</Text>
            <ZdrojoveTema>hlavni</ZdrojoveTema>
            <CiloveTema>uvodniPozdrav</CiloveTema>
            <NastavPriznak JmenoPriznaku="tykani" Prikaz="Inc"
            Hodnota="1" TrvalyPriznak="1" />
      </Zaznam>
      ...
 </SeznamVet>
<SeznamOdpovedi>
```

```
    <Zaznam Id="16" KolikratPouzito="19">
        <Text>Ahoj, jak se máš?</Text>
        <ZdrojoveTema>uvodniPozdrav</ZdrojoveTema>
        <CiloveTema>hlavni</CiloveTema>
        <Podminka Priznak="tykani" Operator="&gt;" Hodnota="0" />
    </Zaznam>
    ...
</SeznamOdpovedi>
```

Věta i odpověď jsou zde definovány jako element *Zaznam*, jejich význam se liší podle toho, jestli je *Zaznam* umístěn v elementu *SeznamOdpovedi* nebo *SeznamVet.* Každý záznam je jednoznačně identifikován atributem *id*, jenž je unikátní jak v rámci elementu *SeznamVet,* tak v rámci elementu *SeznamOdpovedi*, tj. v celém souboru je vždy jen jeden *Zaznam* se stejným *id.*

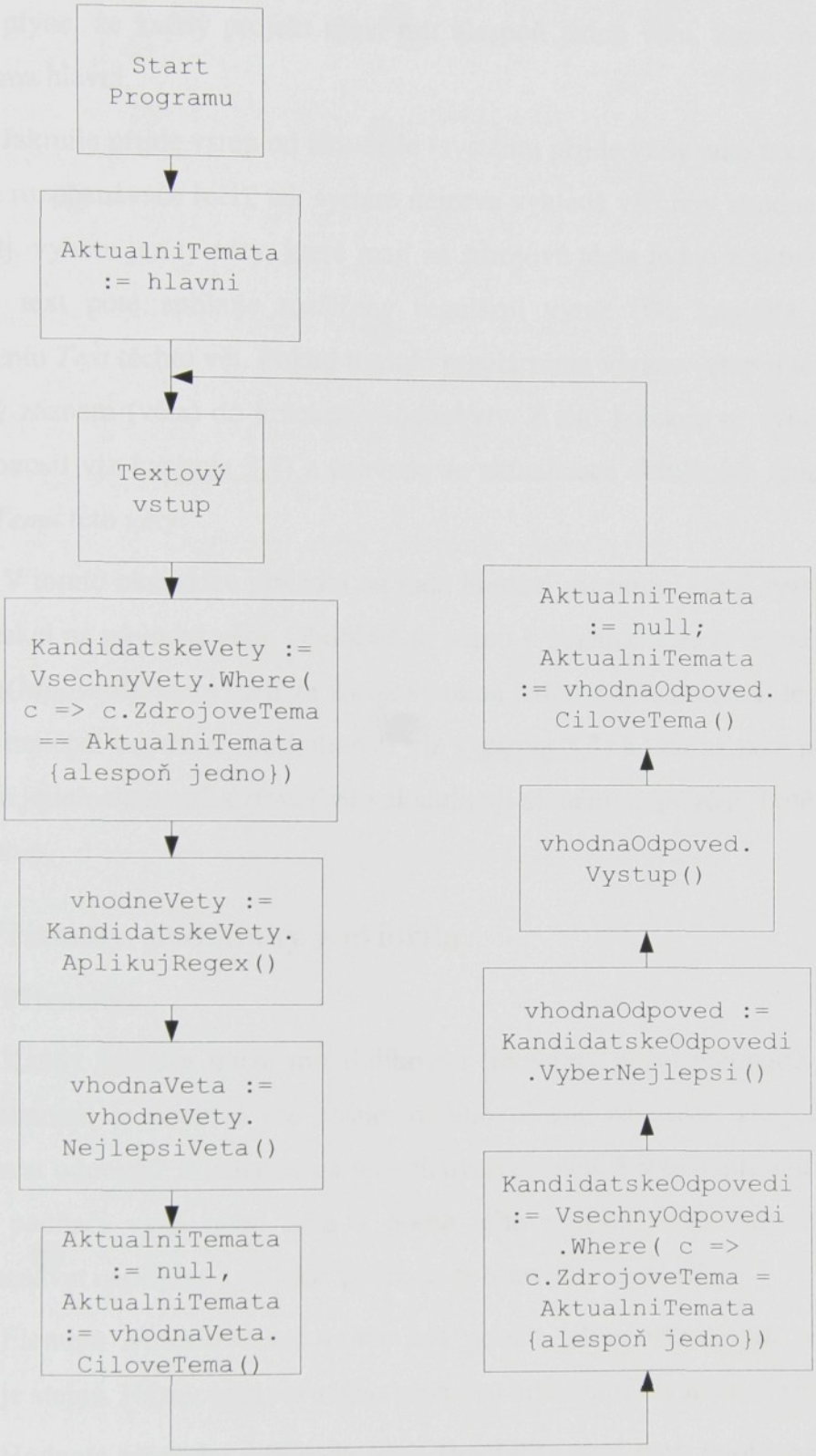
Každý záznam má definován element *ZdrojoveTema.* Těchto elementů může být pro každou větu (odpověď) definováno libovolné množství, minimálně pak jeden. Podle tohoto elementu se zjišťuje, jestli je daná věta (odpověď) v danou chvíli využitelná.

Element *Text* je v případě, že se jedná o *Zaznam* umístěný v elementu *SeznamVet* rozšířený regulární výraz (viz kapitola 3.6.1), který se aplikuje na vstupní sekvenci znaků od uživatele. Pokud se jedná o odpověď, je obsah elementu *Text* věta, jež má systém říci, pokud je tato odpověď použita.

Element *CiloveTema* udává, jakým směrem se má komunikace přesunout v případě, že byla tato věta využita. Těchto elementů může být opět definováno více, komunikace se může v jednom okamžiku nacházet ve více tématech.

Další elementy a atributy nejsou v tuto chvíli důležité, je jim věnována samostatná kapitola 3.2.

Vlastní systém výběru vhodných témat je zobrazen na obr.2. Pro stručnost je zde používána syntax rozšiřujících metod a celý systém je poněkud zjednodušen, zároveň některá jména proměnných plně neodpovídají skutečnosti.



Obrázek 2: Rozhodovací mechanismus

Program si v každém okamžiku udržuje interní kolekci aktuálních témat *AktualniTemata*. Jelikož po startu programu by byla tato kolekce prázdná a nemohla by se rozpoznat žádná věta, určil jsem, že po startu programu se do ní přidá téma *hlavni*.

Z toho plyne, že každý projekt musí mít alespoň jednu větu, která má jako zdrojové téma téma hlavní.

Jakmile přijde vstup od uživatele (systému přijde vždy jako text, i když se jedná o text z rozpoznávače řeči), tak systém nejprve vyhledá všechny vhodné věty z hlediska témat, tj. vybere jen ty věty, které mají za zdrojové téma jedno z aktuálních témat. Na vstupní text poté aplikuje rozšířený regulární výraz (viz kapitola 3.6.1) uvedený v elementu *Text* těchto vět. Pokud tomuto regulárnímu výrazu vstupní text vyhoví, přidá se daný záznam (věta) do kolekce *vhodneVety*. Z této kolekce se vybere nejlepší věta (podrobnosti viz kapitola 3.4) a provede se aktualizace aktuálních témat dle elementů *CiloveTema* této věty.

V tomto okamžiku přichází na řadu hledání záznamu, který systém použije jako svou reakci na původní větu. Obdobně se berou v úvahu pouze ty záznamy (v elementu *SeznamOdpovedi*), které mají za zdrojové téma jedno z aktuálních. Z těchto záznamů se vybere nejlepší odpověď (podrobnosti viz kapitola 3.3) a tato se také použije. Zároveň se podle jejich elementů *CiloveTema* aktualizují aktuální odpovědi. Poté se opět čeká na další vstup.

3.2 Příznaky, podmínky a priorita

3.2.1 Příznaky

Každý záznam může mít definován libovolný počet elementů *NastavPriznak*. Tento element se používá pro nastavení libovolného příznaku, který nám může dále zpřesňovat odpověď založenou na tematických celcích. Může například zaznamenat, že člověk počítači vyká nebo tyká a podle toho upřesnit svou odpověď. Může také zaznamenávat například „náladu" počítače či libovolný další údaj.

Element *NastavPriznak* může být použit jak v otázce, tak v odpovědi, jeho funkce je stejná. Název upravovaného příznaku udává atribut *JmenoPriznaku*.

Hodnota příznaku z důvodu větší flexibility není typu boolean ale int, takže se vždy s každým nastavením každého příznaku musí uvést také operace, která se má provést oproti současné hodnotě tohoto příznaku. Tato operace je specifikována atributem *Prikaz*, jehož hodnoty mohou být *Equal* (přiřazení), *Dec* (dekrementace) nebo *Inc* (inkrementace). Atribut *Hodnota* udává, o kolik se má hodnota příznaku změnit (v případě operaci *Inc* nebo *Dec*) nebo na jakou hodnotu se má příznak nastavit.

Posledním nepopsaným atributem v tomto elementu je atribut *TrvalyPriznak*. Tento atribut může mít hodnotu buď 1 (pravda) nebo 0 (nepravda) a specifikuje, jaká je doba života tohoto příznaku. V případě, že má hodnotu 1, tak příznak je platný navždy, (případně do té doby, než nějaký jiný záznam změní u tohoto příznaku životnost), v případě, že má hodnotu 0, platí příznak jen jeden komunikační cyklus (tj. 1x otázka, 1x odpověď), poté se příznak vymaže včetně své hodnoty).

V případě, že se nastavovaný příznak ještě během komunikace nezaznamenal a přijde instrukce k jeho zvětšení nebo zmenšení, je mu přiřazena výchozí hodnota 0 a poté je vykonána daná instrukce.

Element *NastavPriznak* může konkrétně vypadat takto:

```
<NastavPriznak JmenoPriznaku="tykani" Prikaz="Inc" Hodnota="1"
TrvalyPriznak="1" />
```

V tomto případě se nastavuje příznak „*tykani*" (atribut *JmenoPriznaku*), hodnota tohoto příznaku se zvýší (atribut *Inc*) o hodnotu 1 (atribut *Hodnota*). Tento příznak bude platný až do konce komunikace.

3.2.2 Podmínky

Nastavování příznaků by nemělo opodstatnění, pokud by se s nimi dále nepracovalo. Tyto příznaky mohou být vyhodnocovány a podle nich upřesněna odpověď. Můžeme tak zadat, že daná odpověď se použije pouze v tom případě, že hodnota určeného příznaku je v rámci námi definovaných mezí.

K tomuto vymezení se používá volitelný element *Podmínka*, který může být použit v odpovědi (v otázce nikoli). V případě použití tohoto elementu musíme kromě názvu příznaku, který chceme vyhodnotit (atribut *Priznak*), specifikovat také hodnotu, oproti které chceme aktuální hodnotu příznaku porovnávat (atribut *Hodnota*), a také porovnávací operátor (atribut *Operator*).

Tento element může být konkrétně použit například následujícím způsobem:

```
<Podminka Priznak="tykani" Operator="&gt;" Hodnota="0" />
```

V tomto případě se odpověď, ve které je umístěn tento element, použije pouze v tom případě, že aktuální hodnota příznaku *tykani* (atribut *Priznak*) je větší (atribut *Operator*) než 0 (atribut *hodnota*).

Atribut operátor může mít hodnotu jakéhokoliv ze standardních operátorů =, <, >, <=, >=. U porovnávacích operátorů se musí použít XML odkaz na entitu, aby tyto operátory nemohli být zaměněny s XML značkováním.

3.2.3 Priorita

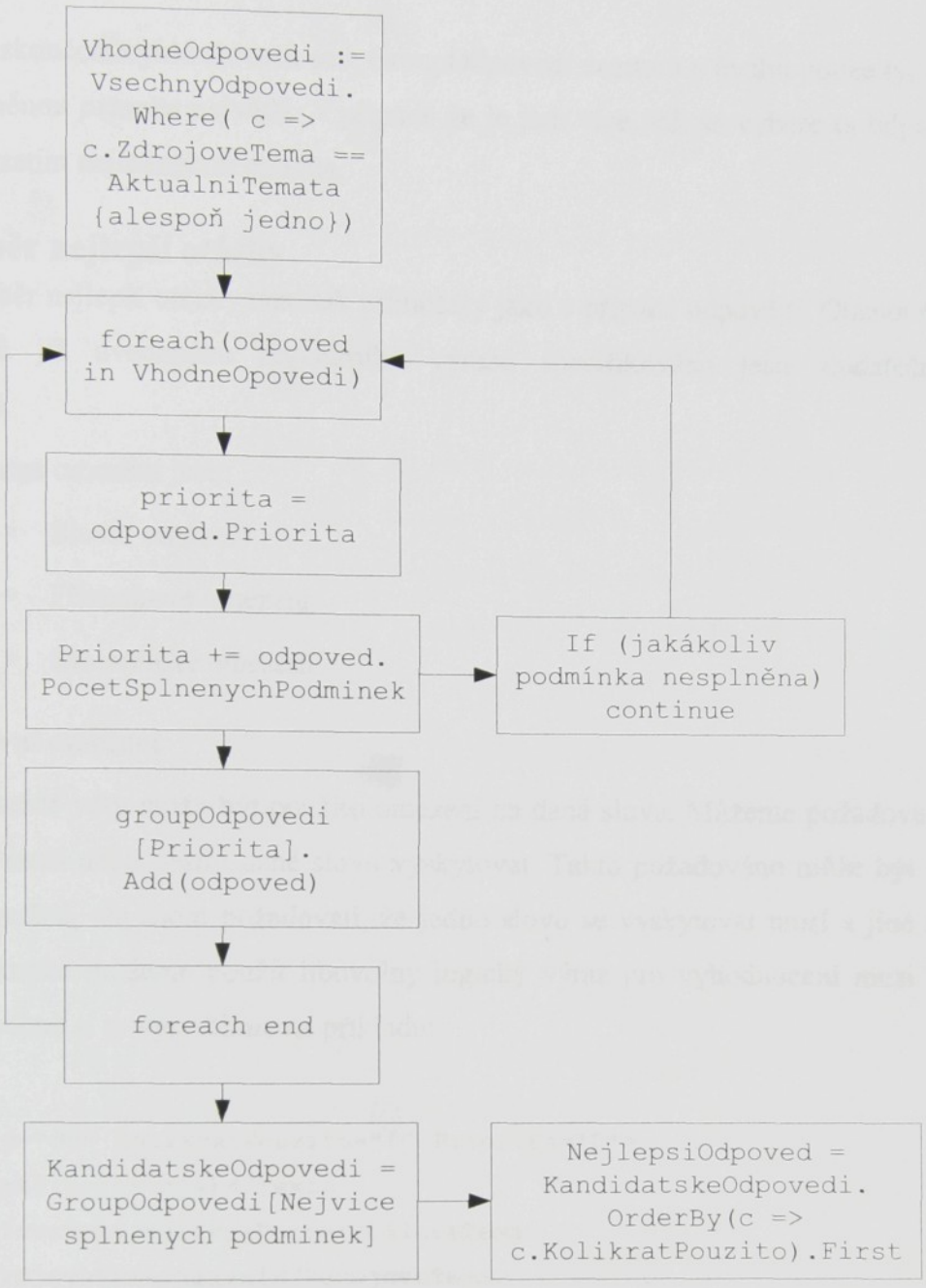
Každý záznam může mít specifikovánu i svou prioritu a může tak být upřednostněn před jinými. Priorita se specifikuje jako atribut u každého záznamu a její hodnota může být mezi 0 až 99.

```
<Zaznam Id="18" KolikratPouzito="0" Priorita="6">
...
</Zaznam>
```

Priorita se však chová u otázek a u odpovědí poněkud odlišně. U otázek je priorita absolutní a vždy „vyhrává" priorita vyšší, u odpovědí se ještě bere v úvahu počet slněných podmínek. Podrobnosti viz následující dvě podkapitoly.

3.3 Výběr nejlepší odpovědi

Nejlepší odpověď se ze všech možných vybírá na základě počtu splněných podmínek a udané priority. V prvé řadě se vezmou všechny odpovědi, které přicházejí v danou chvíli v úvahu dle tématu a poté se všechny zkoušejí analyzovat. Mechanismus výběru nejlepší odpovědi zobrazuje obr. č. 3:



Obrázek 3: Mechanismus výběru nejlepší odpovědi

Po vybrání všech odpovědí, které přicházejí v úvahu dle aktuálního tématu, se všechny odpovědi postupně procházejí cyklem foreach. Pro každou odpověď se spočítá pomocná proměnná *priorita* jako součet atributu *priorita* uvedeného u každé odpovědi a počtu splněných podmínek.

Pokud je jakákoliv podmínka nesplněna (to může nastat buď v případě, že byla podmínka opravdu nesplněna, anebo v případě, že se nenašel příznak, který podmínka vyžadovala), tak se daná odpověď vyřadí, respektive se nepřidá do kolekce *groupOdpovedi*. V této kolekci jsou jednotlivé odpovědi uloženy v závislosti na tom, jakou hodnotu měla proměnná *priorita*.

Po skončení cyklu foreach se z GroupOdpovedi vezmou v úvahu pouze ty, které měly proměnou *priorita* největší. V případě že je jich více, tak se vybere ta odpověď, která byla zatím nejméněkrát použita.

3.4 Výběr nejlepší otázky

Výběr nejlepší otázky není tak přímočarý jako v případě odpovědí. Otázka může být kromě již uvedeného regulárního výrazu specifikována ještě dodatečnými omezeními.

Možná omezení jsou:

- Slovní omezení
- Příznakové omezení
- Sémantické omezení

3.4.1 Slovní omezení

U každé věty může být použito omezení na daná slova. Můžeme požadovat, že se ve větě musí nebo nesmí dané slovo vyskytovat. Takto požadováno může být více slov a je možné, abychom požadovali, že jedno slovo se vyskytovat musí a jiné zase nesmí. Dokonce můžeme použít libovolný logický výraz pro vyhodnocení mezi více slovy. Podrobně si to vysvětlíme na příkladu:

```
<Zaznam Id="99" KolikratPouzito="0" Priorita="0">
      <Text>(kolik\sje)</Text>
      <CiloveTema>kolikJePocty</CiloveTema>
      <ZdrojoveTema>hlavni</ZdrojoveTema>
      <OmezeniSlov>
            <Slovo Cislo="1" Text="plus" />
            <Slovo Cislo="2" Text="mínus" />
            <Slovo Cislo="3" Text="krát" />
            <Slovo Cislo="4" Text="lomeno" />
            <Slovo Cislo="5" Text="děleno" />
            <Slovo Cislo="6" Text="hodin" />
            <Formule>(1 OR 2 OR 3 OR 4 OR 5) AND NOT 6 </Formule>
      </OmezeniSlov>
</Zaznam>
```

Jednotlivá slovní omezení jsou definována volitelným elementem *OmezeniSlov*. Tento element může obsahovat libovolný počet dětských elementů *slovo* a musí obsahovat jeden element *formule*.

Element *slovo* obsahuje dva atributy; atribut *Text* specifikuje hledané slovo a atribut *Cislo* slouží jako identifikátor daného slova pro vyhodnocení. Obsahem atributu *Text* může být kromě prostého textu také libovolný rozšířený regulární výraz (viz kapitola 3.6.1).

Element *Formule* obsahuje výraz, který se vyhodnocuje a podle kterého se určuje, jestli je slovní omezení splněno. V tomto elementu můžeme používat čísla, která jsme použili jako označení pro jednotlivá slova a logické operátory NOT, AND a OR. Tato čísla jsou použity jako odkazy na výsledek dotazu, zda se dané slovo ve vstupním textu vyskytlo či ne.

V tomto případě je slovní omezení splněno v případě, že se ve vstupním textu vyskytlo alespoň jedno slovo ze slov zastupených čísly 1-5 a zároveň se nevyskytlo slovo zastoupené číslem 6.

3.4.2 Příznakové omezení

Ve slovníku (viz kapitola 6.5), který je připojen k projektu, je možné specifikovat kromě různých morfologických tvarů ke každému slovu také příznak. Na tento příznak je možné aplikovat omezení podobně jako na normální slova. Tento element se v tomto případě jmenuje *OmezeniPriznaku*.

```
<Zaznam Id="90" KolikratPouzito="0" Priorita="0">
      <Text>(jmenuj[iu])</Text>
      <ZdrojoveTema>hlavni</ZdrojoveTema>
      <CiloveTema>predstaveniZnam</CiloveTema>

      <OmezeniPriznaku>
            <Formule>1</Formule>
            <Priznak Cislo="1" Text="vlastnijmeno" />
      </OmezeniPriznaku>
</Zaznam>
```

Element *Formule* zde má naprosto stejnou funkci jako v případě slovních omezení, pouze se zde jednotlivá čísla odkazují na příznaky.

V atributu *Text* v elementu *Priznak* je definován vlastní příznak, tento příznak se však vyhledává u základního tvaru slova ve slovníku.

V tomto případě se daná věta použije v případě, když je splněn základní regulární výraz a zároveň je ve vstupním textu nalezeno slovo, které je označeno příznakem *vlastniJmeno*.

3.4.3 Sémantické omezení

Každý projekt musí obsahovat také soubor se základními sémantickými vazbami (viz kapitola 6.4). Rozšíření programu tímto způsobem umožňuje programu zareagovat i v případě, že nezná přímo dané slovo, ale ví, jak zareagovat na slovo které s ním souvisí, respektive zná slovo, které je obecnější než zadané.

Sémantické omezení rozpoznávané v programu zná pouze jednu vazbu, a to je specializace.

```
<schema namespace="onto">
  <objekt Jmeno="ATOM" />
  <objekt Jmeno="věc" JeSpecializaci="ATOM" />

  <objekt Jmeno="potrava" JeSpecializaci="věc" />
  <objekt Jmeno="pití" JeSpecializaci="potrava" />
  <objekt Jmeno="jídlo" JeSpecializaci="potrava" />

  <objekt Jmeno="alkohol" JeSpecializaci="pití" />
  <objekt Jmeno="guláš" JeSpecializaci="jídlo" />
 </schema>
```

V tomto případě je definován základní prvek ATOM a od toho jsou postupně odvozovány další prvky, takže například slovo guláš je specializací jídla, to je specializací potravy a tak dále.

Pokud v programu chceme reagovat na to, že uživatel by si dal něco k jídlu nebo k pití, stačí vyžadovat výskyt sémantického prvku potrava a v případě, že je sémantický slovník dostatečně obsáhlý, tak jsme zároveň postihli velké množství požadavků na konkrétní jídlo.

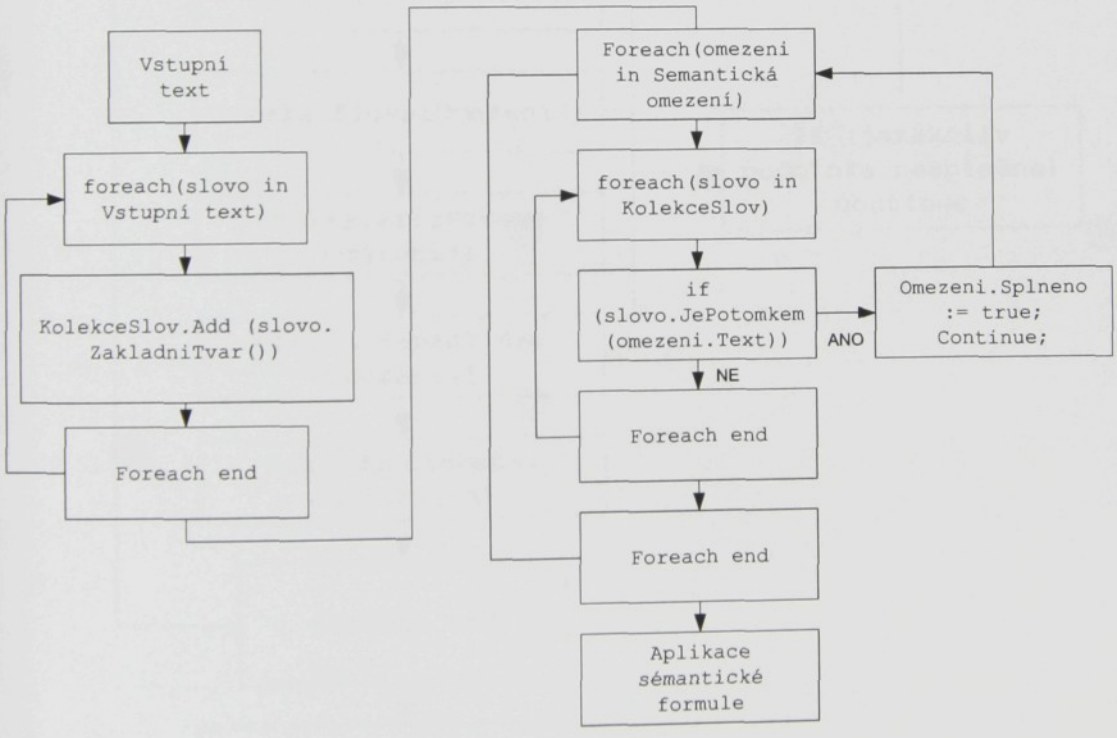
Pro tento příklad bychom si omezení sémantiky mohli vymezit tímto způsobem:

```
<Zaznam Id="119" KolikratPouzito="0" Priorita="0">
      <Text>({dal})</Text>
      <ZdrojoveTema>hlavni</ZdrojoveTema>
      <CiloveTema>chtitJistPit</CiloveTema>
```

```
    <OmezeniSemantiky>
        <Formule>1</Formule>
        <Priznak Cislo="1" Text="jídlo" />
    </OmezeniSemantiky>
</Zaznam>
```

Jednotlivé elementy mají obdobný význam jako u ostatních omezení, klíčový je element *Priznak,* který vymezuje jednotlivé sémantické omezení.

V tomto případě se na vstupní text zkouší nejprve aplikovat rozšířený regulární výraz ({dal}) a poté se zkouší aplikovat sémantické omezení. Vyhodnocovací mechanismus funguje tak, že se nejprve projdou všechna slova ve vstupním textu a najde se k nim pomocí slovníku (viz kapitola 6.5) základní tvar (v případě, že ve slovníku nejsou, se použijí v tom tvaru, v jakém byly použity). Poté se pro každé sémantické omezení tato slova procházejí a v případě, že toto slovo je specializací slova uvedeného v příslušném atributu *Text*, je toto sémantické omezení splněno. Tento algoritmus je zobrazen na obr. č. 4.

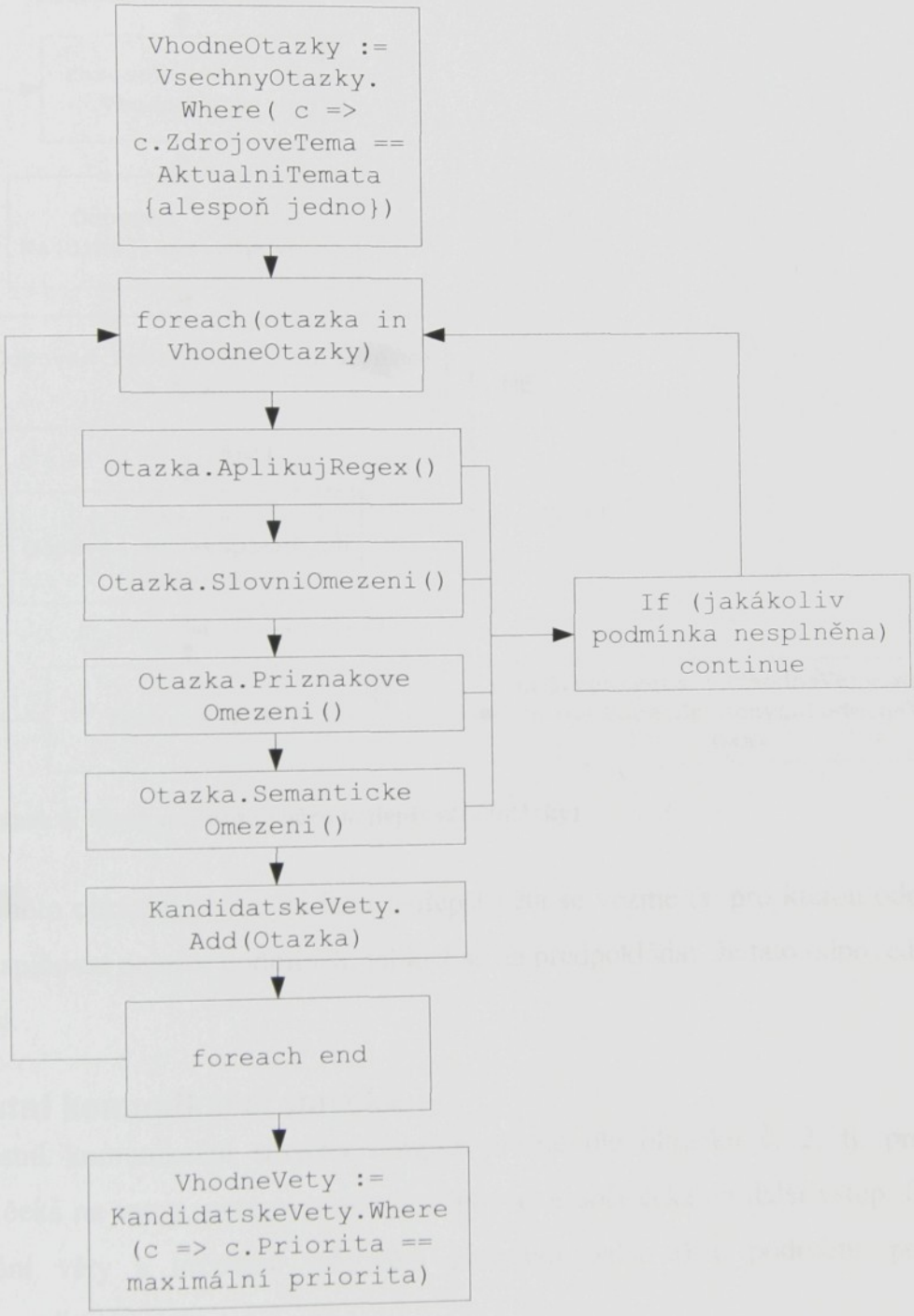


Obrázek 4: Algoritmus vyhodnocování sémantických formulí

3.4.4 Vlastní výběr nejlepší otázky

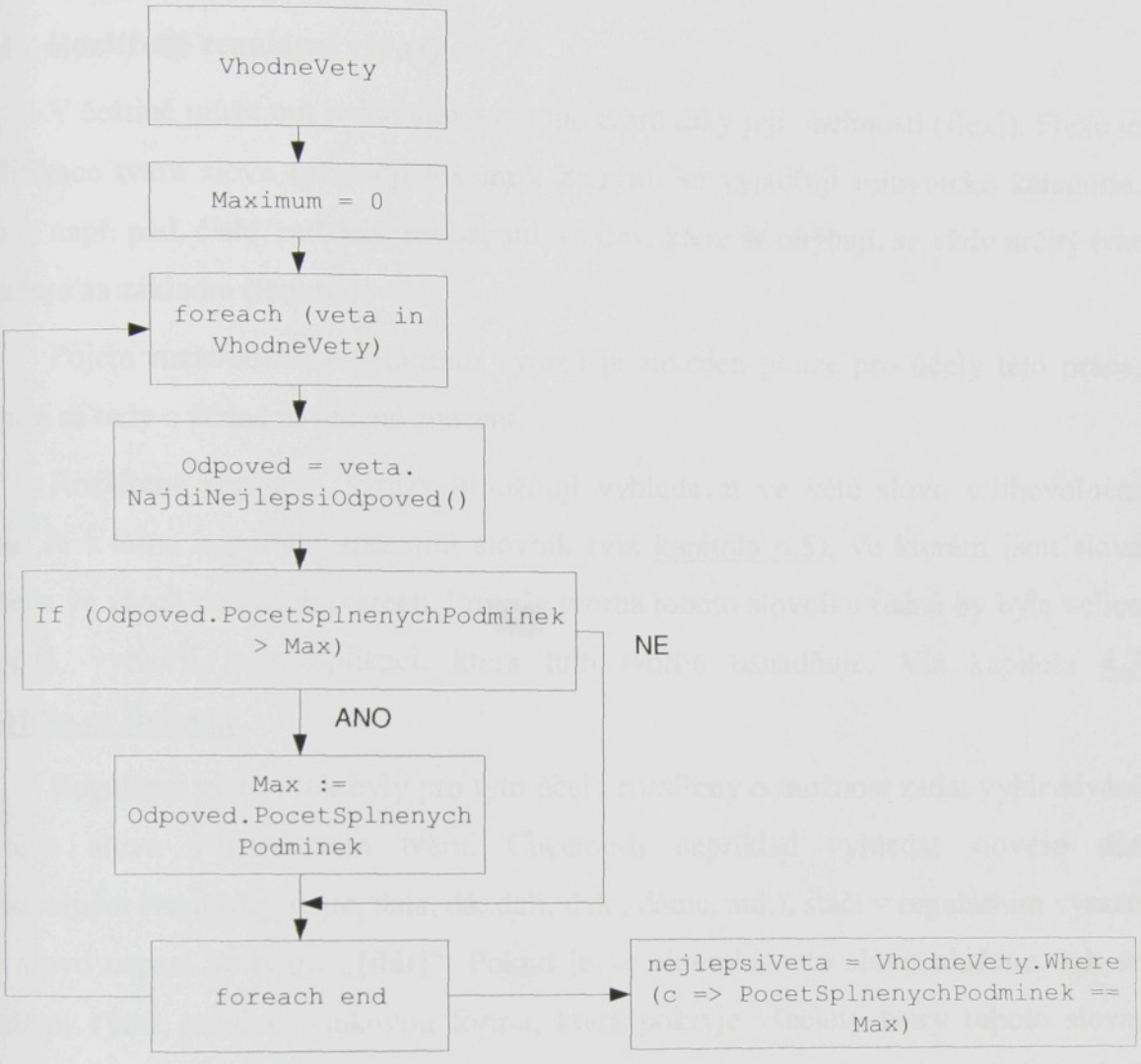
Vlastní výběr nejlepší otázky se dá rozdělit do dvou kroků:

1) Nejprve se vyberou vhodné věty na základě aktuálních témat a poté se postupně aplikují regulární výraz a jednotlivá omezení (slovní, příznakové a sémantické). Věty, které splňují všechna omezení, se přidají do seznamu *Kandidátských vět*. Z kandidátských vět se posléze vyberou jen ty, jejichž priorita je největší (zjistí se největší použitá priorita a vyberou se pouze věty s touto prioritou), a dají se do kolekce vhodné věty. Tento výběr je podrobně zobrazen na obrázku č.5:



Obrázek 5: Mechanismus výběru vhodných otázek

2) Nyní máme v kolekci *Vhodné věty* jednu nebo více vět, které splňují všechna omezení. Mechanismus výběru jedné konkrétní otázky, která se použije, je na obrázku č. 6:



Obrázek 6: Mechanismus výběru nejlepší věty (otázky)

Z tohoto obrázku plyne, že jako nejlepší věta se vezme ta, pro kterou odpověď na ní bude splňovat nejvíce podmínek, jelikož se dá předpokládat, že tato odpověď bude nejpřesnější.

3.5 Vlastní komunikační smyčka

Vlastní komunikační smyčka funguje přesně dle obrázku č. 2, tj. program sekvenčně čeká na vstup, poté vygeneruje odpověď a opět čeká na další vstup. Během rozpoznávání věty a určování odpovědi probíhají další akce podrobně popsané v minulých podkapitolách.

3.6 Rozšířené možnosti vstupu a výstupu

Vlastní program umožňuje ještě několik dodatečných vylepšení. Jedná se zejména o Rozšířené regulární výrazy a podporu výstupních skriptů.

3.6.1 Rozšířené regulární výrazy

V češtině může mít jedno slovo mnoho tvarů díky její ohebnosti (flexi). Flexe je modifikace tvarů slova (přesněji lexému), kterými se vyjadřují mluvnické kategorie, jako je např. pád, číslo, rod, čas, osoba, atd. U slov, která se ohýbají, se vždy určitý tvar považuje za základní (lemma).

Pojem rozšířeného regulárního výrazu je zaveden pouze pro účely této práce, neopírá se tedy o žádné zavedené značení.

Rozšířené regulární výrazy umožňují vyhledávat ve větě slovo v libovolném tvaru. Je k tomu zapotřebí speciální slovník (viz kapitola 6.5), ve kterém jsou slova uložena ve všech možných tvarech. Protože tvorba tohoto slovníku ručně by byla velice náročná, vytvořil jsem aplikaci, která tuto tvorbu usnadňuje. Viz kapitola **4.3 Pravidla.cz Dolovač**.

Regulární výrazy tak byly pro tyto účely rozšířeny o možnost zadat vyhledávání určitého slova v libovolném tvaru. Chceme-li například vyhledat sloveso **dát** v libovolném tvaru (dej, dejte, dala, dá, dali, dalo, dáme, atd.), stačí v regulárním výrazu toto slovo napsat ve tvaru „**{dát}**“. Pokud je ve slovníku toto slovo uloženo, tak se regulární výraz rozvine v takovou formu, která pokryje všechny tvary tohoto slova. Pokud toto slovo ve slovníku není, použije se regulární výraz s odstraněnými složenými závorkami.

Toto je praktické zejména z toho důvodu, že z rozpoznávače řeči dost často získáme slovo, které se liší jen koncovkou.

3.6.2 Výstupní skripty

Jelikož v zadání práce bylo, aby byl celý systém co nejvíce obecný, řešil jsem problém jak umožnit virtuální osobnosti být v kontaktu s okolním světem (myšleno s počítačem na kterém je spuštěn). Bez tohoto kontaktu by virtuální osobnost nedokázala například ani odpovědět na otázku kolik je hodin.

Vyřešil jsem to voláním externích skriptů, takže aplikace může být rozšiřována o nové možnosti, aniž by se muselo zasahovat do jejich zdrojových kódů. Využil jsem

k tomu projekt CS-Script (viz [18]), který umožňuje volat funkce ze zdrojových souborů v jazyce C# tak, jako by to byly funkce v nějakém skriptovacím jazyce.

Všechny tyto funkce musí být umístěny v podadresáři *Skripty* v souboru *VystupniSkripty.cs*. Jednotlivé funkce musí být definovány jako statické a musejí vracet hodnotu typu *String*. Parametry nesmí mít žádné.

V tomto skriptovacím souboru je například naprogramována funkce, která vrací aktuální čas, nebo funkce, která zjišťuje aktuální počasí v ČR z internetu. Možností je skutečně mnoho, jelikož v těchto skriptech lze používat všechny možnosti jako v běžných programech napsaných v jazyce C#.

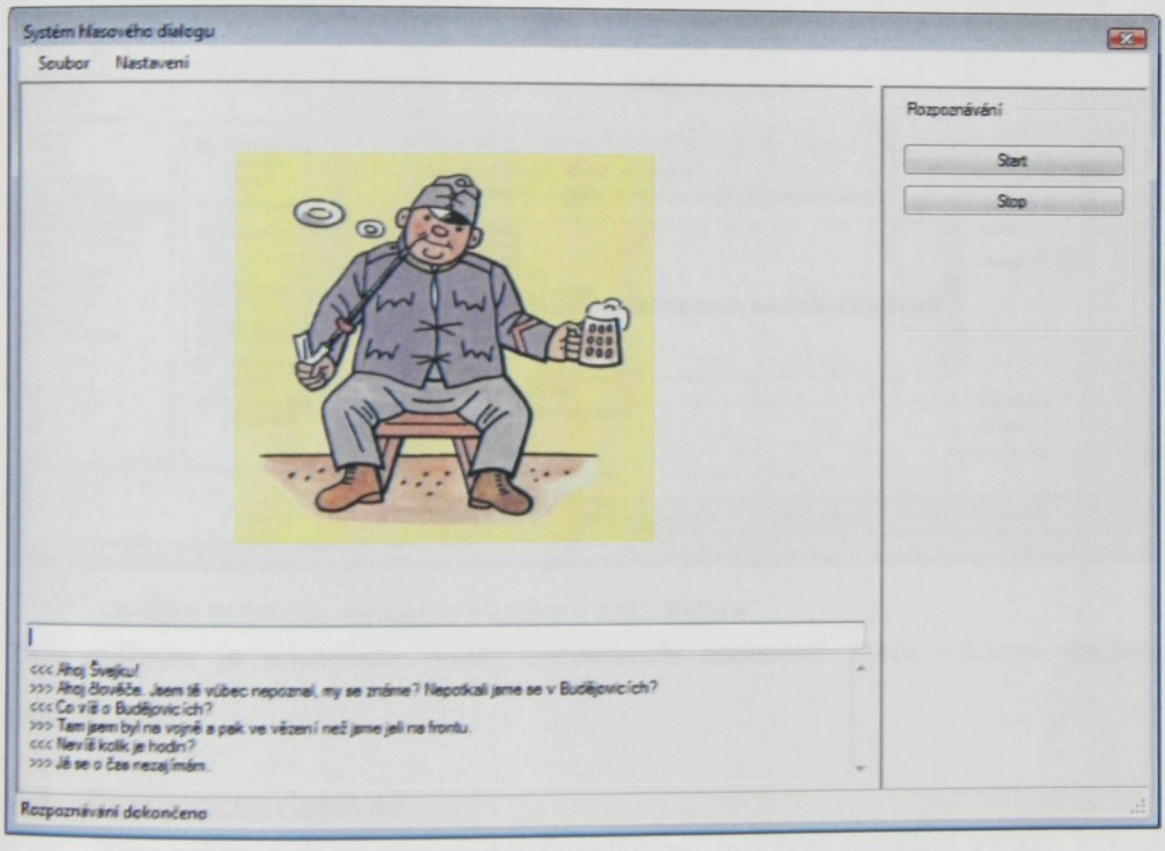
Funkce, která zjišťuje aktuální čas, je v tomto souboru definována pod názvem **PresnyCas**. Pokud chceme výsledek této funkce použít, stačí (v odpovědi) zapsat název funkce v tomto tvaru: **{@PresnyCas}**. Zapisuje se tedy jako název funkce ve složených závorkách spolu s uvozujícím znakem **@**. V tomto případě by celý tvar odpovědi nejspíš vypadal jako:

```
<Text>Právě je {@PresnyCas}.</Text>
```

4 Naprogramované aplikace

4.1 Systém hlasového dialogu

Vlastní aplikace, skrze kterou se komunikuje s virtuální osobností, vyžaduje ke svému běhu tak jako všechny ostatní aplikace nainstalovaný .NET Framework 3.5. Aplikace je inicializována konfiguračním souborem *HlasovyDialog.exe.config*. V tomto souboru je v prvé řadě uveden projekt, který se má načíst, je zde také specifikována cesta ke knihovnám syntezátoru a jsou zde volby pro konfiguraci syntezátoru a rozpoznávače včetně volby, kdy se úplně zakáže jejich použití. V tomto případě pak program funguje pouze v textovém režimu. Snímek obrazovky z vlastní aplikace je na obrázku číslo 7.



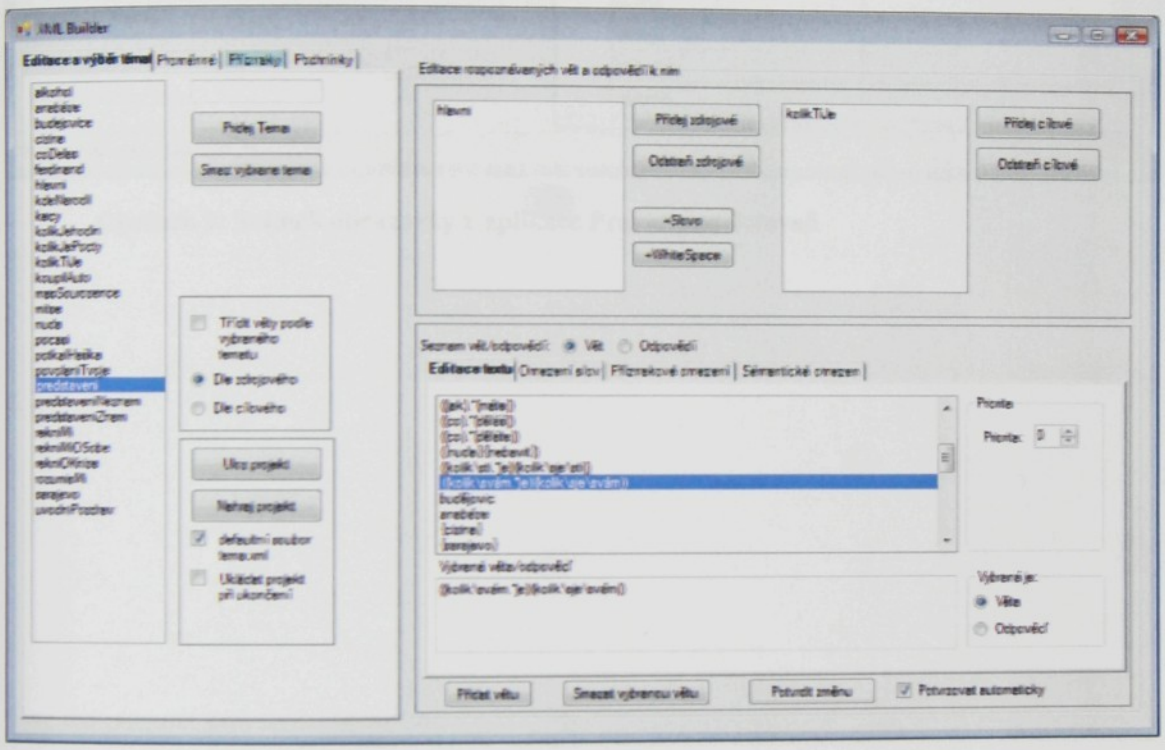
Obrázek 7:Snímek obrazovky z aplikace Systém hlasového dialogu

Osobnost, která s člověkem bude komunikovat včetně ilustračního obrázku, je nastavena v konfiguračním souboru projektu (viz kapitola 6.2).

4.2 XML Builder

Jelikož psát celé projekty ručně za použití běžného textového editoru by bylo velice náročné na čas a náchylné na chyby, je součástí této práce také program, který umožňuje vyrobit celý projekt s virtuální osobností bez znalosti vnitřní struktury aplikace pouze za použití myši a vyplňování připravených polí. Tento program podstatně zkracuje čas nutný na vyrobení jednotlivých virtuálních osobností.

S pomocí tohoto programu lze vytvořit projekt včetně všech rozšiřujících možností, jako jsou podmínky, slovní a sémantická omezení apod. Jsou zde navíc barevně zvýrazněny položky, které již jsou vyplněny, takže editace již hotových osobností je přehlednější.

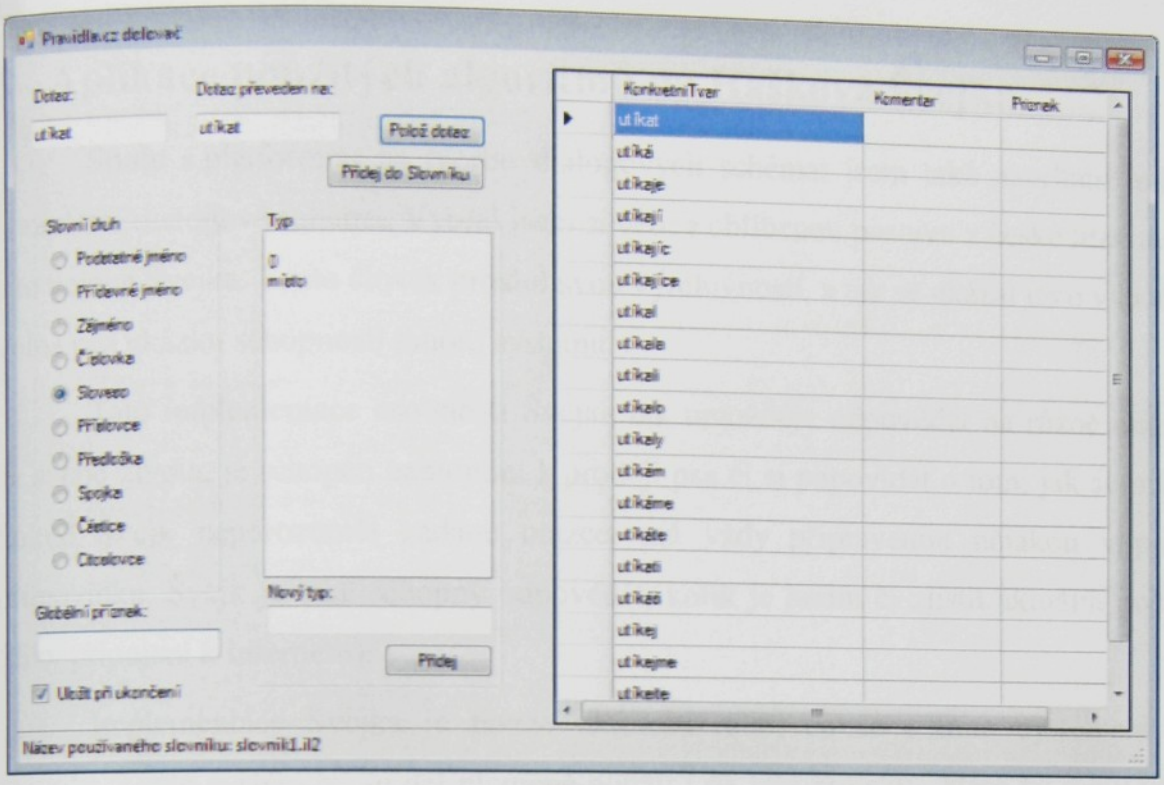


Obrázek 8: Snímek obrazovky z aplikace XML Builder

Tato aplikace je z hlediska tvorby virtuálních osobností spolu s hlavní aplikací nejdůležitější.

4.3 Pravidla.cz dolovač

Tato aplikace slouží pro tvorbu slovníků pro rozšířené regulární výrazy. Kromě prosté editace a ručního zadání všech tvarů slov umožňuje také napojení na server www.pravidla.cz, na kterém je snadné vyhledat k jednomu zadanému slovu všechny jeho možné tvary. Tímto způsobem je možné značně urychlit tvorbu uživatelských slovníků.



Obrázek 9: Snímek obrazovky z aplikace Pravidla.cz dolovač

5 Aplikace použitých algoritmů na Haškova Švejka

Spolu s platformou na tvorbu dialogových schémat jsem také navrhnul jedno kompletní dialogové schéma. Vybral jsem si velice oblíbenou postavu z české literatury, a to vojáka Švejka. Tento člověk proslul svou výmluvností, a tak se ukázal jako výborná volba pro ukázku schopností tohoto systému.

Tato implementace osobnosti Švejka tak umožňuje odpovědět na různé otázky ze svého života, je schopen nabídnout k prodeji psa či si popovídat o tom, jak se máte. Pokud Švejk neporozuměl zadané otázce, má vždy připravenou nějakou vtipnou průpovídku. Švejk je také schopný odpovědět, kolik je hodin či zjistit aktuální počasí (díky připojení k internetu).

Implementace Švejka je pouze ukázkou toho, co se s tímto systémem dá vytvořit. Tento systém by se dal nicméně použít i na jiné projekty, které by měly větší ambice. Dal by se například implementovat virtuální učitel, který by fungoval jako mluvící encyklopedie a odpovídal by na zadané otázky. V takovémto rozhovoru by se také více uplatnilo použití témat, jelikož by se dala hesla neustále více upřesňovat.

6 Formáty používaných souborů

6.1 Soubor s větami virtuální osobnosti

Vlastní soubor s větami virtuální osobnosti je tak jako většina ostatních souborů ve formátu XML. Jeho přesnou strukturu udává XML schéma uvedené jako příloha č. 1.

Tento soubor obsahuje 3 klíčové elementy. Dva z nich již byly podrobně popsány v předchozích kapitolách (elementy *SeznamVet* a *SeznamOdpovedi*). Element, o kterém jsme se ještě nezmínili, je *SeznamTemat*.

V tomto elementu je seznam názvů všech použitých témat. Jednotlivá témata jsou uvedena v elementu *tema* v tomto tvaru:

```
<Tema Nazev="hlavni" />
```

Celý element *SeznamTemat* by nebyl nezbytně nutný, jelikož informace které obsahuje, by se daly získat jednoduchým XPath dotazem. Je zde hlavně z důvodu snazší ruční editace v případě, že by byla potřeba.

6.2 Konfigurační soubor HlasovyDialog.exe.config

Tento soubor je standardní konfigurační soubor .NET aplikace a je tudíž také ve formátu XML. Popišme si jednotlivé volby, které jdou v tomto souboru nastavit.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="cestaKSyntezatoru" value="E:\synt\Synt" />
        <add key="HlasovyVystup" value="True" />
        <add key="HlasovyVstup" value="False" />
        <!--    Defaultní hodnoty-->
        <add key="VystupPovolen" value="True" />
        <add key="VstupPovolen" value="True" />
        <add key="JmenoProjektu" value="ProjektSvejk.xml" />
    </appSettings>
</configuration>
```

Volba *cestaKSyntezatoru* specifikuje cestu k souborům syntezátoru. V případě, že je syntéza zapnuta, je třeba, aby cesta ukazovala skutečně k syntezátoru, jinak program skončí chybou.

Volba *HlasovyVystup* udává, jestli se má inicializovat syntezátor a obdobně volba *HlasovyVstup* udává, jestli se má inicializovat rozpoznávač řeči. Pokud jsou tyto volby vypnuty, je možné program spustit i na počítači, kde není rozpoznávač, respektive syntezátor, nainstalován. V případě, že jsou tyto volby vypnuty, již nelze za běhu programu dodatečně inicializovat zvukový vstup/výstup.

Volby *VystupPovolen* a *VstupPovolen* jsou předvolby pro vlastní program. I když je rozpoznávání řeči nebo syntéza iniciována, je možné jejich používání vypnout a používat pouze textový vstup a výstup. V případě potřeby je pak lze během programu kdykoli zapnout.

Volba *JmenoProjektu* udává umístění vlastního souboru s projektem. Obsah tohoto souboru je podrobně popsán v následující kapitole.

6.3 Soubor s projektem

V tomto souboru je specifikováno umístění všech souborů, které vytvářejí virtuální osobnost. Jeho tvar je následující:

```
<?xml version="1.0" encoding="utf-8" ?>
<konfigurace>
      <Temata Jmeno="temata.xml" />
      <Slovnik Jmeno="slovnik1.il2" />
      <Semantika Jmeno="onto.xml" />
      <Obrazek Jmeno="svejk2.jpg" />
</konfigurace>
```

Význam jednotlivých elementů je:

- *Temata* – soubor s větami popsaný v kapitole 6.1.
- *Slovnik* – soubor, ve kterém jsou umístěny různé tvary slov. Tento soubor je popsán v kapitole 6.5.
- *Semantika* – soubor se sémantickými vazbami popsaný v kapitole 6.4.
- *Obrazek* – soubor s obrázkem, který bude zobrazen v hlavním okně po startu programu.

Jednotlivé položky mohou obsahovat i cestu k uvedenému souboru, takže tyto soubory mohou být umístěny i v různých adresářích.

6.4 Soubor se sémantickými vazbami

Tento soubor již byl částečně popsán v kapitole 3.4.3. Jelikož struktura tohoto souboru je velice jednoduchá, nebyla pro tento soubor navrhnuta žádná speciální aplikace, která by ho pomáhala naplnit.

Navrhnuté třídy použité v programu umožňují kromě popsaného vztahu *Je specializací* ještě jeden vztah *Je částí*. Ukažme si to na příkladu:

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema namespace="onto">
  <objekt Jmeno="ATOM" />
  <objekt Jmeno="věc" JeSpecializaci="ATOM" />
  <objekt Jmeno="nemovitost" JeSpecializaci="věc" />
  <objekt Jmeno="dům" JeSpecializaci="nemovitost" />
  <objekt Jmeno="pokoj" JeSpecializaci="nemovitost" JeCasti="dům" />
  <objekt Jmeno="okno" JeSpecializaci="nemovitost" JeCasti="dům" />
  <objekt Jmeno="okno" JeSpecializaci="nemovitost" JeCasti="pokoj" />
</schema>
```

Kromě již popsaných atributů se v elementu objekt objevil atribut *JeCasti*. Tento atribut je volitelný a umožňuje specifikovat, že popisovaná věc je částí nějakého objektu. V uvedeném případě tak je například popsáno, že pokoj je částí domu.

Jedna věc může být částí více objektů, v tom případě ji ale musíme uvést vícekrát. V uvedeném příkladě je okno částí jak domu, tak pokoje.

Ačkoli používané třídy umožňují využití i tohoto vztahu, vlastní program ho nevyužívá.

6.5 Soubor se slovníkem

Soubor se slovníkem, kde jsou uvedeny jednotlivé morfologické tvary slov, není jako jediný ve formátu XML. Je to z toho důvodu, že jsem chtěl využít již existující systém Czech Free Morphology (viz [19]). Systém sice nakonec nebyl nasazen, ale byl použit původní formát slovníku. Přesný formát slovníku je k nalezení na stránkách autora [20], jeho kompletní popis by vydal na několik stran textu. Já možnosti tohoto formátu využívám jen nepatrně, v podstatě jsem se omezil na dvojice konkrétního tvaru slova – lemma.

Oproti autorovi jsem však využil komentářovou sekci jednotlivých položek na označení slovních příznaků (viz kapitola 3.4.2).

Vlastní záznam v mém podání vypadá většinou takto:

R|0|jirko|jiří|0|0|0|0|0|0#vlastnijmeno

Písmeno R a znaky 0 se znaky | můžeme považovat za předepsaný formát souboru, pro nás zajímavé položky jsou na jiných místech. Slovo *jirko* je v tomto případě konkrétní tvar slova a následující slovo *jiří* je jeho vzor (lemma). Za poslední nulou je znakem # oddělen vlastní příznak k tomuto slovu, tedy příznak, který nám v tomto případě udává, že se jedná o vlastní jméno. Těchto příznaků může být uvedeno více, oddělují se přitom vždy dalším znakem #.

Všechna slova v tomto slovníku by měla být psána pouze s pomocí malých znaků.

7 Závěr

Cílem této práce bylo navrhnout systém hlasového dialogu s virtuální osobností. Navržený systém toto umožňuje a není přitom omezený jen na jednu konkrétní osobnost, ale je zde ponechána možnost tvorby libovolného počtu virtuálních osobností bez zásahů do zdrojových kódů programu.

Vlastní chování virtuální osobnosti je určeno dialogovými schématy, kdy má k různým otázkám systém připraveny různé odpovědi. Rozhodování o tom, jaká se použije odpověď, je také závislé na průběhu předchozího rozhovoru.

Dialogové schéma je uloženo jako několik XML souborů. Soubor se slovníkem s různými morfologickými tvary slov je uložen v textové podobě. Tvorba dialogových schémat nespočívá ve vytváření těchto souborů ručně v textovém editoru, ale byly navrženy podpůrné aplikace, které umožňují vytvořit kompletní dialogové schéma jen za použití myši a vyplňování připravených textových polí.

Za hlavní přínos mého řešení považuji to, že je zde zapracována jednoduchá sémantika, takže virtuální osobnost si umí spojit, že k sobě patří určité pojmy. Pokud tak například řekneme, že bychom měli chuť na polévku, virtuální osobnost umí na tuto otázku zareagovat, i když má připravenou odpověď jen na podobnou otázku, ve které je místo polévky použito slovo jídlo. Toto je možné jen díky tomu, že je k projektu připojen soubor, ve kterém jsou uvedeny základní sémantické vazby mezi slovy.

Odpovědi virtuální osobnosti se také mohou dynamicky generovat díky rozšiřujícím skriptům, takže je možné se osobnosti například zeptat na aktuální počasí, přičemž se daný skript připojí na internet a stáhne si aktuální stav počasí z internetu. Tímto způsobem se dá virtuální osobnost rozšířit o mnoho zajímavých schopností.

Pokud bych chtěl tento systém srovnat s velice úspěšným anglicky konverzujícím programem ALICE (tento program celkem 3x získal Loebnerovu bronzovou plaketu, což je ocenění pro nejlepší konverzační program), tak moje řešení má obdobné schopnosti jako tento program. Navržený systém je oproti ALICI lépe použitelný pro český jazyk, jelikož respektuje flexi slov, a také respektuje napojení na rozpoznávač řeči, takže jeho vstupem mohou být gramaticky a morfologicky nesprávné věty.

Pokud bychom však srovnali moje řešení včetně všech dialogových schémat a současnou verzi ALICE, tak bude mít ALICE lepší reakce. Je to dáno tím, že ALICE obsahuje přes 40000 šablon a s tím moje projekty nemohou soupeřit. Hlavně tímto směrem bych viděl možnost dalšího pokračování v tomto projektu.

Bylo vytvořeno jedno kompletní dialogové schéma, které se snaží simulovat postavu Švejka z knih o Dobrém vojáku Švejkovi od Jaroslava Haška.

8 Seznam použité literatury

[1] WEIZENBAUM, Joseph. ELIZA – *A Computer Program For the Study of Natural Language Communication Between Man and Machine*, 1966. 21.2.2008. http://i5.nyu.edu/~mm64/x52.9265/january1966.html

[2] COLBY, Kenneth. *PARRY: Paranoia mental hospital patient*, 23.2.2008. http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/classics/parry/0.html

[3] ALICE A.I. Foundation. *Alicebot*, 11.12.2007. http://alicebot.blogspot.com

[4] InteliWISE. *InteliWISE AVATAR*, 23.2.2008. http://www.inteliwise.com/

[5] Zabaware, Inc. *Ultra Hal Assistant*, 24.2.2008. http://www.zabaware.com/assistant/index.htm

[6] JANDA, Aleš. *IQ Pokyd*, 25.2.2008. http://iqpokyd.kyblsoft.cz/

[7] ANTONIČ, Mišo. *Ludvík*, 26.3.2008. http://www.ludvik.sk/

[8] POLAKOVIČ, Jaroslav. *WinKec*, 24.4.2008. http://winkec.wz.cz/

[9] InfraDrive. *RoboMatic X1*, 23.3.2008. http://www.infradrive.com/robomatic.php

[10] Speechlab. *Rozmluva s virtuální osobností – Projekt Švejk*, 2002. 25.4.2008. http://itakura.kes.vslib.cz/kes/svejk.html

[11] Bill Chamberlain. *Getting a computer to write about itself*, 28.4.2008. http://www.atariarchives.org/deli/write_about_itself.php

[12] NOUZA, Jan. *Počítačové zpracování řeči: cíle, problémy, metody a aplikace.* Sborník článků. První vydání. Liberec, Technická univerzita v Liberci, Fakulta mechatroniky a mezioborových inženýrských studií, 2001. 122 stran. ISBN 80-7083-551-6.

[13] MOUČEK, Roman, KONOPÍK, Miloslav. *Sémantika v dialogových systémech*, 3.5.2008.

http://hilbert.chtf.stuba.sk/KUZV/download/kuzv-moucek-konopik.pdf

[14] NEJEDLOVÁ, Dana. *Tvorba slovníků a jazykových modelů pro automatický přepis zpravodajských pořadů*, 3.5.2008.

www.fm.tul.cz/files/autoreferat_nejedlova.pdf

[15] HUANG X., ACERO A., HON H. W..: *Spoken Language Processing. A Guide to Theory, Algorithm and System Development*. New Jersey. Prentice Hall, 2001. 960 stran. ISBN: 0130226165.

[16] BRADLEY, Neil. *XML: Kompletní průvodce*. První vydání. Praha 7, Grada Publishing, 2000. 540 stran. ISBN 80-7169-949-7.

[17] MARGUERIE, Fabrice. LINQ in Action, 10.5.2008.

http://linqinaction.net/

[18] SHILO, Oleg. *CS-Script – The C# Script Engine*, 12.5.2008.

http://www.members.optusnet.com.au/~olegshilo/

[19] HAJIČ, Jan. *Czech "Free" Morphology*. 2001. 10.5.2008.
http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Morphology/index.html#dict

[20] HAJIČ, Jan. *Morphology and Tagging*. 2001. 10.5.2008.
http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/index.html

[21] HAJIC, Jan. *Disambiguation of Rich Inflection. (Computational Morphology of Czech)*. První vydání. Praha. Karolinum, 2004. 328 stran. ISBN 80-246-0282-2.

[22] Wintellect. *Power Collections for .NET*. 8.4.2008.

http://www.wintellect.com/PowerCollections.aspx

[23] HAŠEK, Jaroslav. *Osudy dobrého vojáka Švejka za světové války*. Třetí vydání. Praha. Práce, 1955. 768 stran.

9 Příloha 1 – xml schéma souboru s větami

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Projekt">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="SeznamTemat">
     <xs:complexType>
      <xs:sequence>
       <xs:element maxOccurs="unbounded" name="Tema">
        <xs:complexType>
         <xs:attribute name="Nazev" type="xs:string" use="required" />
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="SeznamVet">
     <xs:complexType>
      <xs:sequence>
       <xs:element maxOccurs="unbounded" name="Zaznam">
        <xs:complexType>
         <xs:sequence>
          <xs:choice maxOccurs="unbounded">
           <xs:element name="Text" type="xs:string" />
           <xs:element name="ZdrojoveTema" type="xs:string" />
           <xs:element name="NactiPromennou" type="xs:string" />
           <xs:element name="CiloveTema" type="xs:string" />
           <xs:element maxOccurs="unbounded" name="NastavPriznak">
            <xs:complexType>
             <xs:attribute name="JmenoPriznaku" type="xs:string" use="required" />
             <xs:attribute name="Prikaz" type="xs:string" use="required" />
             <xs:attribute name="Hodnota" type="xs:unsignedByte" use="required" />
             <xs:attribute name="TrvalyPriznak" type="xs:unsignedByte" use="required" />
            </xs:complexType>
           </xs:element>
           <xs:element name="OmezeniSlov">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="Formule" type="xs:string" />
              <xs:element maxOccurs="unbounded" name="Slovo">
               <xs:complexType>
                <xs:attribute name="Cislo" type="xs:unsignedByte" use="required" />
                <xs:attribute name="Text" type="xs:string" use="required" />
               </xs:complexType>
              </xs:element>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="OmezeniPriznaku">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="Formule" type="xs:string" />
```

```
            <xs:element name="Priznak">
             <xs:complexType>
              <xs:attribute name="Cislo" type="xs:unsignedByte" use="required" />
              <xs:attribute name="Text" type="xs:string" use="required" />
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
         <xs:element name="OmezeniSemantiky">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Formule" type="xs:unsignedByte" />
            <xs:element name="Priznak">
             <xs:complexType>
              <xs:attribute name="Cislo" type="xs:unsignedByte" use="required" />
              <xs:attribute name="Text" type="xs:string" use="required" />
             </xs:complexType>
            </xs:element>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:choice>
       </xs:sequence>
       <xs:attribute name="Id" type="xs:unsignedByte" use="required" />
       <xs:attribute name="KolikratPouzito" type="xs:unsignedByte" use="required" />
       <xs:attribute name="Priorita" type="xs:unsignedByte" use="required" />
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="SeznamOdpovedi">
   <xs:complexType>
    <xs:sequence>
     <xs:element maxOccurs="unbounded" name="Zaznam">
      <xs:complexType>
       <xs:sequence>
        <xs:choice maxOccurs="unbounded">
         <xs:element name="Text" type="xs:string" />
         <xs:element name="ZdrojoveTema" type="xs:string" />
         <xs:element name="CiloveTema" type="xs:string" />
         <xs:element maxOccurs="unbounded" name="Podminka">
          <xs:complexType>
           <xs:attribute name="Priznak" type="xs:string" use="required" />
           <xs:attribute name="Operator" type="xs:string" use="required" />
           <xs:attribute name="Hodnota" type="xs:unsignedByte" use="required" />
          </xs:complexType>
         </xs:element>
        </xs:choice>
       </xs:sequence>
       <xs:attribute name="Id" type="xs:unsignedByte" use="required" />
       <xs:attribute name="KolikratPouzito" type="xs:unsignedByte" use="required" />
       <xs:attribute name="Priorita" type="xs:unsignedByte" use="optional" />
      </xs:complexType>
     </xs:element>
    </xs:sequence>
```

```
    </xs:complexType>
   </xs:element>
   <xs:element name="SeznamPromennych">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Promenna">
       <xs:complexType>
        <xs:attribute name="Nazev" type="xs:string" use="required" />
        <xs:attribute name="ZakladniTvar" type="xs:string" use="required" />
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="Nazev" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
</xs:schema>
```