TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: Informační technologie

Distribuovaný systém pro rozpoznávání spojité řeči

Distributed system for continuous speech recognizing

Diplomová práce

Autor: **Bc. Miroslav Jasso**
Vedoucí práce: Ing. Jindřich Žďánský, Ph.D.

V Liberci 20.5.2009

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Ústav informačních technologií a elektroniky
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Miroslav JASSO**

Studijní program: **N2612 Elektrotechnika a informatika**

Studijní obor: **Informační technologie**

Název tématu: **Distribuovaný systém pro rozpoznávání spojité řeči**

Zásady pro vypracování:

1) Seznamte se s technologií v2t vyvinutou v laboratoři Speechlab
2) Navrhněte distribuovaný systém pro rozpoznávání spojité řeči přes Internet
3) Ověřte funkčnost navrženéjo řešení ve zkušebním provozu

Rozsah grafických prací: **Dle potřeby dokumentace**
Rozsah pracovní zprávy: **cca 40 - 50 stran**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Huang, X., Acero, A., Hon, H-W.: Spoken Language Processing

Vedoucí diplomové práce: **Ing. Jindřich Žďánský, Ph.D.**
Ústav informačních technologií a elektroniky

Datum zadání diplomové práce: **31. října 2008**
Termín odevzdání diplomové práce: **29. května 2009**

V. Kopecký
prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Ing. Ondřej Ne
vedoucí úst

V Liberci dne 31. října 2008

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 26.5.09

Podpis M. Jonáš

Anotace

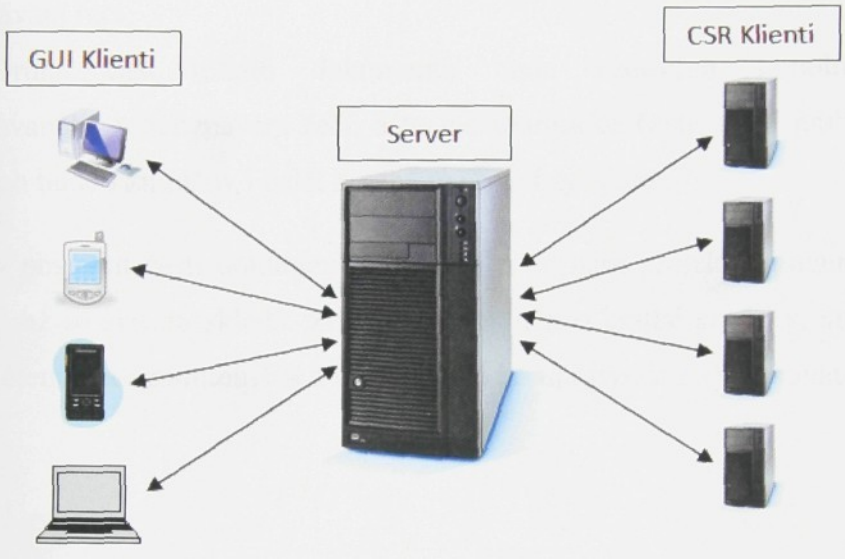
Cílem této diplomové práce je navrhnout a naprogramovat systém distribuovaného rozpoznávání řeči pomocí Rozpoznávače spojité řeči, který vyvíjí ústav ITE. Za tímto účelem bude navrženo několik odlišných principů komunikace a jeden bude realizován v praxi v podobě funkčního systému. Systém se bude skládat ze tří samostatných aplikací. Hlavní součástí bude server. Dále dva klienti, jeden grafický určený pro diktování a zobrazování rozpoznaného textu, druhý klient bude tvořit mezičlánek mezi Rozpoznávačem spojité řeči a serverem.

Annotation

The aim of this diploma thesis is to suggest and preprogram the system of distribution of the speech by recognizer of connected speech which is developed by ITE institution. It will be designed some different communication principles and one of them will be realized in use in the representating functional system. The system will consist of three independent applications. The main component is server. Then two clients, one of them will be used for dictation and representing recognized text, the second one will be a connecting link between recognizer of the fluent speech and server.

1 Úvod

Cílem této diplomové práce je navrhnout distribuovaný systém typu klient-server, který bude schopen v reálném čase provádět automatický přepis mluveného slova do textové podoby za využití technologie v2t vyvíjené laboratoří SpeechLab na Technické Univerzitě v Liberci. Seznámit se s vybranými technologiemi komunikace přes internet, kde bude kladen důraz hlavně na minimalizaci datového toku a robustnosti z hlediska výpadků v síti a uvedení jedné z těchto technologií do praxe.



Obrázek 1: systém distribuovaného rozpoznávání řeči

Výsledkem práce bude funkční systém distribuovaného rozpoznávání složený ze tří komponent (viz obrázek 1). Hlavní z komponent bude Server distribuovaného rozpoznávání, který bude mít za úkol propojení dvou klientů. Grafický klient (GUI) bude určený pro diktování a zobrazování rozpoznaného textu. Tento klient, resp. jeho verze pro různá zařízení a operační systémy, by měl být schopen fungovat na jakémkoliv zařízení disponujícím nahráváním zvuku a internetovým připojením. Druhý klient (CSR) bude tvořit mezičlánek mezi aplikací Rozpoznávače spojité řeči, která má vlastní komunikační rozhraní a Serverem.

Jedním z vedlejších cílů bude možnost komprese zvukových dat přenášených systémem pro dosažení minimálního datového toku, ovšem s důrazem na zachování kvality zvukových dat.

Toto zadání vzniklo z požadavku na možnost přepisování mluveného slova do textové podoby na pomalejších počítačích a mobilních zařízeních. Samotná aplikace pro rozpoznávání spojité řeči vyvíjená laboratoří SpeechLab potřebuje pro svůj běh výkonné počítače disponující dvou-jádrovými procesory a minimálně 512Mb paměti.

Z toho důvodu bude počítač s nainstalovaným CSR klientem velmi výkonný, avšak u zařízení s GUI klientem není nutné klást důraz na výkon. Díky tomu je možné GUI klienta naprogramovat například i pro mobilní telefon nebo PDA.

V první části této práce, resp. kapitolách dva až čtyři, se budeme zabývat teoretickou stránkou, a to převážně způsoby komunikace přes internet, respektive mezi klienty navzájem. Dále se budeme věnovat problematice VoIP (přenos hlasu pomocí protokolu IP) a kompresnímu kodeku Speex, který bude využit také v systému spojitého rozpoznávání řeči.

Druhá část tohoto dokumentu bude věnována samotnému systému distribuovaného rozpoznávání řeči, a to jak teoretické části, tak i praktické. V těchto kapitolách budou zmíněny i další možné rozšíření systému.

V poslední části dokumentu je umístěn seznam projektů, jmenných prostorů a tříd, z nichž se systém skládá. Seznam je doplněn o krátké popisky, aby byl zdrojový kód pro čtenáře srozumitelnější a bylo snadné ho upravovat a vylepšovat.

2 Windows sockets (WinSock)

2.1 Popis

Windows Sockets API, zkráceně WinSock, je technická specifikace, která definuje, jak mají aplikace pod Windows využívající síťové připojení přistupovat k těmto službám. Převážně se pak jedná o možnost přistupovat k síti pomocí protokolu IP. Definuje standardní rozhraní mezi aplikací a IP protokolem. Názvosloví Winsock čerpá z Berkeley sockets, resp. přidává prefix WSA a také neblokující volání některých funkcí.

Obrázek 2: logo implementace windows sockets

2.2 Historie

Dřívější operační systémy Microsoftu (MS-DOS i Windows) nabízely aplikacím přístup pouze k protokolům relační vrstvy modelu ISO/OSI, což z hlediska programátora nebylo přílišné ulehčení práce. Teprve od Windows 2.0 začaly snahy o implementaci rozhraní mezi aplikací a TCP/IP protokolem. Avšak WinSock je záležitostí až Windows 3.x, kde ale nebyl nativně podporován. Nicméně ho bylo možné doinstalovat.

První podoba Windows Sockets API byla představena v roce 1991. První revize byla ovšem nárokována mnoha představiteli různých firem (Sun Microsystems, Microsoft, ...), a tak nebylo možné se dohodnout na copyrightu. Díky tomu se první revize téměř nepoužívala a v roce 1993 byla vydána revize 1.1, kterou Microsoft nabízel ve svých Operačních systémech.

2.3 Technologie

Windows Sockets API se skládá ze dvou částí API a SPI. API definuje aplikační rozhraní pro přístup k WinSock. SPI umožňuje síťovým programátorům definovat vlastní protokoly, které pracují nad WinSocks. Jak již bylo řečeno, interface Windows

sockets je založený na BSD sockets. Rozdíl je hlavně v předponě „WSA“ u všech funkcí WinSocks rozhraní a doplnění neblokujících funkcí pro více vláknové aplikace. To se stalo obrovskou výhodou WinSock API oproti jiným API a je to i hlavním důvodem proč se používá dodnes.

2.4 Specifikace

- **Verze 1.0 (červen 1992)** – Byla velice blízko Berkeley Sockets. V jejích specifikacích nebylo definováno, nad jakými protokoly bude fungovat.
- **Verze 1.1 (leden 1993)** – Specifikace oproti verzi 1.0 nebyla téměř žádným zlepšením, ale jelikož byla implementována Microsoftem v jeho operačních systémech, tak se stala standardem.
- **Verze 2.0 (květen 1994)** – Byla zpětně kompatibilní s verzí 1.1. Zavádí se protokolově nezávislé pojmenování funkcí. Oznamování je nově prováděno pomocí událostí.
- **Verze 2.1.0 (leden 1996)** – První veřejně uvolněná verze specifikace WinSock 2.
- **Verze 2.2.0 (květen 1996)** – Odstraněna podpora 16-bitových aplikací.

2.5 Implementace

- **Verze 1.0** – Nebyla nikdy Microsoftem implementována.
- **Verze 1.1** – Byla distribuována jako přídavný balíček, zvaný Wolverine, pro Windows 3.x a Windows for Workgroups. Ve Windows 95 a Windows NT 3.x bylo WinSocks API již nativně nainstalováno.
- **Verze 2.0** – Bylo možné ji doinstalovat do Windows 95. Nativně podporována od Windows 98 a Windows NT 4.0. Od Windows XP přidána podpora IPv6.

3 VoIP

Voice over Internet Protocol, zkáceně VoIP, je technologie, umožňující přenos hlasu v těle IP paketů prostřednictvím protokolu TCP nebo UDP (UDP je používanější). Využívá se při telefonování prostřednictvím počítačové sítě a to jak internetu, tak například firemního intranetu.

3.1 Kódování hlasu

Vzhledem k tomu, že se pro přenos hlasu používá internet, který na mnoha místech nemá dostatečnou propustnost na přenos surových zvukových vzorků, je potřeba zvukové vzorky komprimovat. Za tímto účelem bylo vyvinuto mnoho kodeků (software sloužící k zakódování i dekódování dat). Tyto kodeky mají různá označení (G.711, G.722, G.723, ...). Prefix G označuje, že byl kodek standardizován organizací ITU. Mezi freewarové kodeky patří například SPEEX nebo iLBC.

3.1.1 Nejznámější kodeky

- **G.711** – Nejjednodušší a nejrozšířenější kodek v oblasti přenosu hlasu. Bohužel nepodporuje kompresi a proto vyžaduje rychlejší datové spojení. Vzorkovací frekvence je 8kHz při rozlišení 8 bitů. Vyžaduje tedy rychlost připojení minimálně 64 kbit/s.
- **G.729** – Nejúspornější kodek. Pro přenos zvukových dat potřebuje pouze 8 kbit/s při zachování obdobné kvality jako G.711.
- **SPEEX** – Freewarový kodek vyvinutý speciálně pro VoIP. Viz kapitola 5.

3.2 Protokoly komunikace

Kromě hlasových dat musí VoIP zařízení přenášet taktéž řídící informace (signalizaci, ověření dostupnosti, atd.). Celý tento systém komunikace zahrnuje VoIP protokol. Mezi nejznámější patří H.323 a SIP. Dále existují speciální firemní protokoly, jako například: Skinny (Cisco), HFA (Siemens) nebo IAX2, což je protokol softwarových ústředen Asterisk. Všechny protokoly přenáší hlas stejným způsobem v RTP, ale liší se v režijních zprávách, službách a signalizaci.

Dnes je nejpoužívanějším protokolem H.323, který nelze jednoduše použít v místech, kde probíhá překlad adres (NAT) nebo existuje proxy server. S tímto nemá

problémy nový protokol SIP, který přenáší hlasová data kódovaná do podoby textu, jako například HTTP nebo FTP.

3.2.1 Činnost protokolů

Činnosti, které musí protokol před a během hovoru zajistit.

1. **Lokalizace účastníka** - nalezení spojení s koncovou stanicí
2. **Zjištění stavu účastníka** - zjištění, jestli je účastník schopen relaci navázat (může mít obsazeno, přesměrováno atd.)
3. **Zjištění možností účastníka** - zjištění, jaké jsou možnosti účastníka (typ kodeku, max. přenosová rychlost, audio/video atd.)
4. **Vlastní navázání spojení**
5. **Řízení probíhajícího spojení** - případné změny vlastností v průběhu relace a činnosti spojené s jejím ukončováním

3.2.2 Protokol SIP

Session Initiation Protocol, zkráceně SIP, je VoIP protokol využívaný k přenosu režijních informací internetové telefonie. Využívá UDP port 5060, ale je schopen pracovat i nad protokolem TCP. K přenosu hlasu se využívá protokol RTP, stejně jako v případě H.323. Oproti H.323 je SIP mnohem jednodušší. Vychází z protokolu HTTP a také využívá některé principy SMTP.

3.2.2.1 Metody protokolu

SIP je textově orientovaný protokol a metody (příkazy) se v něm píší velkými písmeny podle HTTP, ze kterého vychází.

- **REGISTER** - registrace účastníka na SIP Proxy serveru
- **INVITE** - zahájení komunikace o plánované nové relaci
- **ACK** - potvrzení zahájení relace
- **CANCEL** - přerušení zahajovaní relace ještě před jejím navázáním
- **BYE** - ukončení relace

3.2.2.2 Návratové kódy

Návratové kódy protokolu také vycházejí z HTTP. Vedle číselného označení mají jednotlivé chyby také textovou verzi například 200 - OK, 100 - Trying, atd. Přesný popis návratových kódů HTTP popisuje kapitola 4.6.

- **1xx - průběh** - krok probíhá bez problémů, ale ještě není ukončen
- **2xx - úspěch** - krok byl ukončen bez problémů
- **3xx - přesměrování** - krok probíhá, ale ještě se v souvislosti s ním něco očekává
- **4xx - chyba klienta** - požadavek je chybný a nemůže být serverem zpracován
- **5xx - chyba serveru** - požadavek je zřejmě v pořádku, ale chyba je na straně serveru
- **6xx - fatální chyba** - zcela fatální chyba, kterou nelze jakkoliv zpracovat

3.3 Nevýhody VoIP

Hlavní nevýhodou technologie VoIP je v dnešní době její výpadkovost v případě špatného připojení k internetu. S tímto problémem se setkávají hlavně obyčejní uživatelé internetu připojení přes WiFi. V případě připojení přes ADSL tento problém téměř neexistuje. Dalším problémem, vlivem špatného připojení, bývá latence. Ta nejčastěji způsobuje ozvěny, zpožděné reakce, atd.

Jednou z nevýhod je také odesílání faxů, kde díky protokolu UDP není zaručeno doručení kompletní zprávy. V případě hlasu to nebývá problém, ale pro komprimovaný obrázek to bývá fatální. Z toho důvodu vznikl protokol T.38, který odesílá fax jako přílohu e-mailu přes TCP/IP protokol.

UDP protokol je další nevýhodou. Vzhledem k tomu, že neobsahuje žádné mechanizmy garance doručení paketu, může docházet k výpadkům hovoru, aniž by se o tom druhá strana dozvěděla. Není ani zaručeno pořadí paketů, takže je možné přijmout později poslaný paket dříve.

3.4 Bezpečnost

Stejně jako se u většiny datových přenosů přes internet nepoužívá šifrování, tak tomu není jinak ani u VoIP technologie. Od roku 2008 se objevuje standard Secure RTP, který umožňuje šifrovat hovor, ale pouze v případě podpory obou koncových zařízení. Dále je možné v uzavřené komunitě, například firmě, používat technologii voice VPN, která funguje jako obyčejná VPN a šifruje pomocí technologie IPSec.

3.5 Využití VoIP v Distribuovaném rozpoznávání spojité řeči

Jak již bylo zmíněno VoIP slouží k přenosu lidského hlasu prostřednictvím internetu. Hlasová data by tedy mohla být posílána mezi GUIKlientem a Serverem distribuovaného rozpoznávání pomocí protokolu VoIP. Ale zde narážíme na několik zásadních problémů.

Asi nejzávažnějším problémem je negarantované doručení paketu k cíly. Tím pádem by mohlo docházet k výpadkům paketů a zhoršení kvality rozpoznávání. Systém rozpoznávání spojité řeči je na tyto výpadky velice náchylný, proto byl také použit protokol TCP/IP, který garantuje doručení paketu.

Dalším problémem VoIP je jeho neprostupnost skrze většinu HTTP Proxy serverů. Z toho důvodu byl v této diplomové práci použit pro komunikaci mezi GUIKlientem a serverem protokol HTTP na portu 80.

Ale i přes tyto potíže by bylo zajímavé využít možností této technologie a upravit systém pro příjem zvukových dat tímto protokolem.

4 HTTP protokol

Hypertext Transfer Protocol, zkráceně HTTP, je internetový protokol navržený primárně k přenosu internetových stránek ve formátu HTML. Pro přenos využívá port 80 protokolu TCP/IP. Verze 1.1 je definována v dokumentu RFC2616.

V základu tento protokol neumožňuje přenášet binární data, ovšem díky rozšíření MIME lze přenášet jakákoliv data, čehož využívá i současná verze tohoto projektu. Využívá se společně s formátem XML, na jehož principu je také částečně založen.

4.1 URL

URL, celým názvem *Uniform Resource Locator*, je řetězec znaků s definovanou strukturou, který slouží k přesné lokalizaci zdrojů informací (ve smyslu dokument nebo služba) na internetu. URL specifikuje protokol, kterým zdroj komunikuje, dále doménové jméno, resp. IP adresu, port. Za lomítkem se udává poloha dokumentu nebo služby na daném serveru.

Např.: http://www.fm.tul.cz/**cs/struktur_studium.**

Některá pole lze vynechat, jako je v tomto příkladě vynechaný port. Nezadaná položka musí být doplněna webovým nebo jiným klientem za defaultní hodnotu.

Dále je možné na konci URL specifikovat parametry dotazu. Parametry dotazu se vkládají za otazník a hodnoty se jim přiřazují rovnítkem.

Např.: http://www.google.com/search?**q=http+protokol+wiki**

4.2 HTTPS

HTTP secure je zabezpečená verze klasického protokolu HTTP. Umožňuje šifrovat data pomocí TSL nebo SSL (asymetrické šifrování). Server na rozdíl od standardního HTTP neposlouchá na portu 80, ale na 443.

4.2.1 Princip standardního HTTPS

Při zahájení komunikace si vymění veřejné klíče, které by obě strany měly ověřit pomocí jiného komunikačního kanálu. Ověření může proběhnout kontrolou hash otisku veřejného klíče u protistrany. Například lze použít princip přenosu důvěry, kdy nám protistrana předá veřejný klíč, digitálně podepsaný nejlépe certifikační autoritou, které

důvěřujeme a jejíž veřejný klíč máme v důvěryhodném úložišti (THAWTE, VeriSign, atd).

4.2.2 Princip HTTP 1.1 aktualizované hlavičky

Druhá, méně používaná, metoda zabezpečení komunikace mezi serverem a klientem. Výhodou je, že nepotřebuje další port na ověření komunikace a podobně. Následuje krátký příklad, jak tato metoda pracuje.

Klient zahájí komunikaci obyčejným dotazem:

```
GET /encrypted-area HTTP/1.1
Host: www.example.com
```

V případě, že mu server vrátí chybu klienta 426 je klient donucen přejít na šifrovanou komunikaci:

```
HTTP/1.1 426 Upgrade Required
Upgrade: TLS/1.0, HTTP/1.1
Connection: Upgrade
```

Takovýto zabezpečený server nelze rozpoznat podle URL, protože se nespecifikuje HTTPS jako přenosový protokol.

4.3 Činnosti protokolu

Protokol funguje způsobem dotaz - odpověď. Klient (např. webový prohlížeč) pošle serveru (resp. na adresu URL) dotaz ve formě čistého textu, obsahujícího označení požadovaného dokumentu, informace o schopnostech prohlížeče apod. Server poté odpoví pomocí několika řádků textu popisujících výsledek dotazu (zda se dokument podařilo najít, jakého typu dokument je atd.), za kterými následují data požadovaného dokumentu.

Vzhledem k tomu, že je HTTP protokol bezstavový, není možné jakkoliv zajistit, aby si server držel informace o klientovy (například přihlašovací údaje pro internetový obchod). K tomuto účelu slouží tzv. Cookies. Cookies jsou uloženy na straně klienta a lze je kdykoliv znovu použít a odeslat s každým dotazem danému serveru. V případě, že server cookies odmítne (například skončila jejich platnost), je klient požádán o aktuální údaje.

4.4 Ukázka komunikace

- Webový klient požaduje obsah URL „http://www.fm.tul.cz/ite“:

```
Get /ite
Host: www.fm.tul.cz
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; cs; rv:1.9.0.5)
Gecko/2008120122 Firefox/3.0.5
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie:
SESSbeaf9fb90f540c06bdb4281a273c6df4=8bf8bb9e5b88daab324e619f54d4e6a5
If-Modified-Since: Mon, 09 Feb 2009 13:33:40 GMT
Cache-Control: max-age=0
```

- A dostává odpověď:

```
HTTP/1.0 200 OK
Date: Mon, 09 Feb 2009 13:33:53 GMT
Server: Apache/2.0.54 (Linux/SUSE)
X-Powered-By: PHP/4.4.0
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Mon, 09 Feb 2009 13:33:53 GMT
Cache-Control: store, no-cache, must-revalidate, post-check=0, pre-
check=0
Content-Length: 7808
Keep-Alive: timeout=15, max=96
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Hlavička odpovědi obsahuje mimo jiné i návratovou hodnotu, v tomto případě 200 OK. Za hlavičkou následuje jeden prázdný řádek a poté tělo požadovaného dokumentu. V našem případě internetová stránka ústavu ITE.

4.5 Dotazovací metody

- **GET** - Požadavek na uvedené URL, včetně doplňkových dat. Výchozí metoda při požadavku na zobrazení internetových stránek, RSS zdrojů apod.
- **HEAD** - To samé jako metoda GET, ale už nepředává data. Poskytne pouze metadata o požadovaném cíli (velikost, typ, datum změny, atd).
- **POST** - Odesílá uživatelská data na server. Používá se například při odesílání údajů na formuláři (jméno, příjmení, adresa, ...). Data může odesílat i metoda GET, ale jen do velikosti 512 bytů. Metoda POST může odesílat libovolně velká data a zároveň je nepřenášet jako součást URL. Data předávaná metodou POST jsou obsažena v HTTP požadavku.
- **PUT** - Nahraje data na server. URL je jméno vytvářeného souboru. Používá se velmi zřídka, pro nahrávání dat na server se běžně používá FTP nebo SCP/SSH.
- **DELETE** - Smaže uvedený dokument ze serveru. Jsou na to potřeba jistá oprávnění stejně jako u metody PUT.
- **TRACE** - Odešle kopii obdrženého požadavku zpět odesílateli. Klient takto může zjistit, co na požadavku mění nebo přidávají servery, kterými požadavek prochází. Lze využít i k detekci Proxy serverů v síti.
- **OPTIONS** - Dotaz na server, jaké podporuje metody.
- **CONNECT** - Spojí se s uvedeným URL přes uvedený port. Používá se při průchodu skrze proxy server pro ustanovení kanálu SSL.

4.6 Kódy odpovědí

V následujících kapitolách je stručný popis odpovědí HTTP serveru na klientův dotaz. Odpověď přichází ve formě textového řetězce, který obsahuje jak číslo chyby, tak její doplňující textový popis.

4.6.1 Informační kategorie (1xx)

- **100 Continue** - Klient může pokračovat v zasílání požadavku.
- **101 Switching Protocols** - Server mění protokol.

4.6.2 Úspěšný dotaz (2xx)

- **200 OK** - Operace proběhla bez chyby, požadavek je úspěšně splněn. Následuje dokument ze zadané URL.
- **201 Created** - Výsledkem požadavku je nově vytvořený objekt na zadané URL.
- **202 Accepted** - Byl přijat asynchronní požadavek. Požadavek byl správně akceptován, odpovídající činnost se však ještě zatím nemusela provést.
- **204 No Content** - Požadavek byl úspěšný, ale jeho výsledkem nejsou žádná data pro klienta.

4.6.3 Přesměrování (3xx)

- **300 Multiple choises** - Požadovaný zdroj se dá získat z několika různých míst. V odpovědi se vrací seznam všech možností.
- **301 Moved Permanently** - Požadovaná adresa URL se trvale přesunula na novou adresu URL. Všechny další odkazy musí použít tuto novou URL.
- **302 Moved Temporarily** - Požadovaná adresa URL se dočasně přesunula na novou adresu URL. Všechny další odkazy mohou používat dosavadní URL.
- **304 Not Modified** - Podmíněný požadavek byl správně zpracován, dokument však od udané doby nebyl modifikován. Lze použít dokument z cash prohlížeče.

4.6.4 Chyba klienta (4xx)

- **400 Bad Request** - Server nerozumí požadavku, klient jej musí opravit a poslat znovu.
- **401 Unauthorized** - Jestliže byl původní požadavek klienta anonymní, musí být nyní autentizován. Pokud už požadavek byl autentizován, pak byl přístup odepřen.
- **403 Forbidden** - Server nemůže požadavku vyhovět, autorizace nebyla úspěšná.
- **404 Not Found** - Server nenašel zadanou adresu URL.
- **405 Method Not Allowed** - Použitá metoda není přípustná pro dosažení požadovaného objektu.
- **406 Not Acceptable** - Požadovaný objekt není k dispozici ve formátu podporovaném klientem.
- **408 Request Timeout** - Klient nedokončil odesílání požadavku v časovém limitu.

- **410 Gone** – Požadovaný objekt byl trvale odstraněn.
- **415 Unsupported Media Type** - Požadavek obsahuje data ve formátu, kterému server nerozumí.

4.6.5 Chyba serveru (5xx)

- **500 Internal Server Error** - Na serveru došlo k neočekávané chybě.
- **501 Not Implemented** - Tento požadavek server nepodporuje.
- **502 Bad Gateway** - Proxy server nebo brána obdržely od dalšího serveru neplatnou odpověď.
- **503 Service Unavailable** - Server dočasně nemůže nebo nechce zpracovat požadavek. Většinou když je přetížený nebo se provádí údržba.
- **505 HTTP Version Not Supported** – Server nepodporuje verzi HTTP zadanou v požadavku.

5 Kodek Speex

Tato kapitola popisuje základní principy VoIP kodeku Speex. Dále bude porovnáno několik módů, v kterých kodek pracuje. Bude kladen důraz hlavně na porovnání kvality hlasu v závislosti na datovém toku. Detailní principy komprese a s ní souvisejících algoritmů je možné najít na internetové adrese http://speex.org/docs/manual/speex-manual/.

Vlajková loď projektu OGG v oblasti VoIP. Speex je open-source kodek (enkodér, dekodér) využívaný hojně ve VoIP technologiích. Většina kodeků dnes využívaných v IP telefonii (G.711, G.729, ...) byla primárně navržena na kompresi hovorů v mobilních telefonech ještě před vznikem VoIP. U Speexu tomu tak nebylo. Proto mu jeho úzká specializace dává značné výhody na poli VoIP. Vzhledem k tomu musí být Speex kodek odolný vůči výpadkům celých paketů, avšak nemusí řešit chyby jednotlivých bitů. To je řešeno již protokolem UDP. Vzhledem k těmto požadavkům bylo možné zvolit kompresní techniku CELP (Code excited linear prediction).

Speex byl navržen pro použití se třemi vzorkovacími frekvencemi 8 kHz (narrowband), 16 kHz (wideband) a 32 kHz (ultra-wideband). Kvalita kódování se nastavuje v intervalu od 0 do 10. Obecně lze říci, že čím je parametr vyšší, tím lepší kvalita zvuku.

5.1 Hlavní výhody kodeku Speex

- Open-source, vyvíjený pod licencí BSD. Nezatížený žádnými patenty.
- Široké rozpětí velikostí datového toku (od 2 kbit/s do 44 kbit/s).
- Dynamická změna bit-rate za chodu, možnost VBR (Variabilní bit-rate).
- Detekce aktivity hlasu (když řečník nemluví, je použitý velmi malý bit-rate).
- Ultra-wideband mód pro 32 kHz (dokonce až 48 kHz).
- Možnost kódování stereo zvuku.

5.2 CELP (Code excited linear prediction)

CELP je algoritmus využívaný pro kompresi lidského hlasu. Původní návrh pochází od M.R.Schroedera a B.S.Atala z roku 1985. Postupem času byl zdokonalován a přizpůsobován novým technologiím. Z něho pochází mnoho dalších podobných algoritmů (např.: ACELP, RCELP, VSELP, ...).

Více informací o algoritmu CELP je možné nalézt na internetové adrese http://en.wikipedia.org/wiki/CELP.

5.2.1 Stavební kameny algoritmu

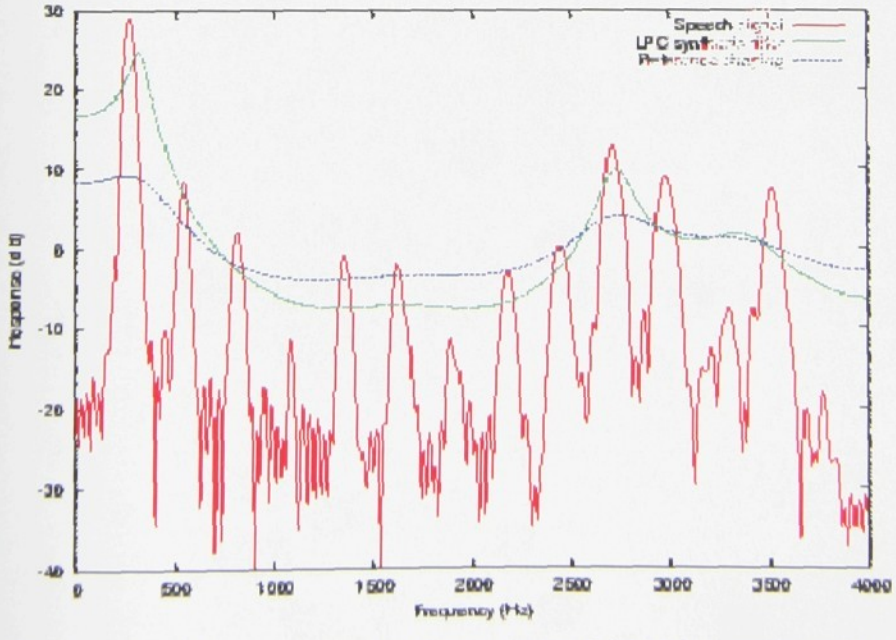
1. Použití „source-filter model of speech production" a „linear prediction" (koeficient LSP). Tento model slouží k upravení zdrojového zvuku tak, aby se důležité frekvence zvýraznily a nedůležité potlačily. Vychází z analýzy lidské řeči a lidského sluchu. Využívá se hlavně v analýze a syntéze lidské řeči.
2. Použití adaptivní a fixované kódovací tabulky jako vstupu pro „linear prediction". Výpočet probíhá podle následujícího vzorce, který udává predikovaný koeficient v závislosti na předchozích změřených koeficientech a koeficientu predikce malé a.

$$\hat{x}(n) = -\sum_{i=1}^{p} a_i x(n-i)$$

3. Zpracování vstupních dat ve smyčce zahrnující obě předchozí akce.
4. Nepovinná operace Vektorové kvantizace (koeficient VQ).

5.2.2 Vyhlazení šumu

CELP, jako většina algoritmů pro kompresi hlasu, umožňuje i redukci a vyhlazení šumu. Na obrázku 3 je možné vidět, jakým způsobem algoritmus šum vyhlazuje.



Obrázek 3: vyhlazení šumu kodekem speex [viz http://speex.org]

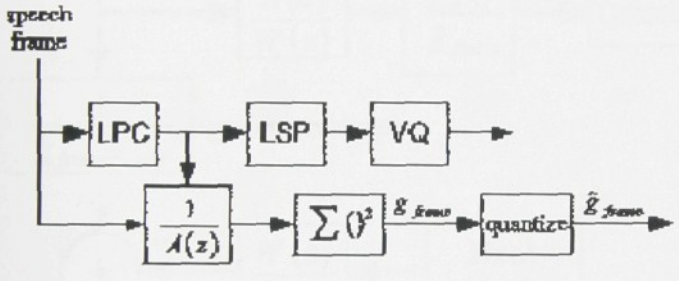
5.3 Speex narrowband mód

V tomto nastavení zpracovává Speex vstupní data o frekvenci 8kHz. Velikost jednoho framu je 20ms řeči, to odpovídá 160 vzorkům. Každý frame je rozdělen na 4 sub-framy, každý o 40 vzorcích.

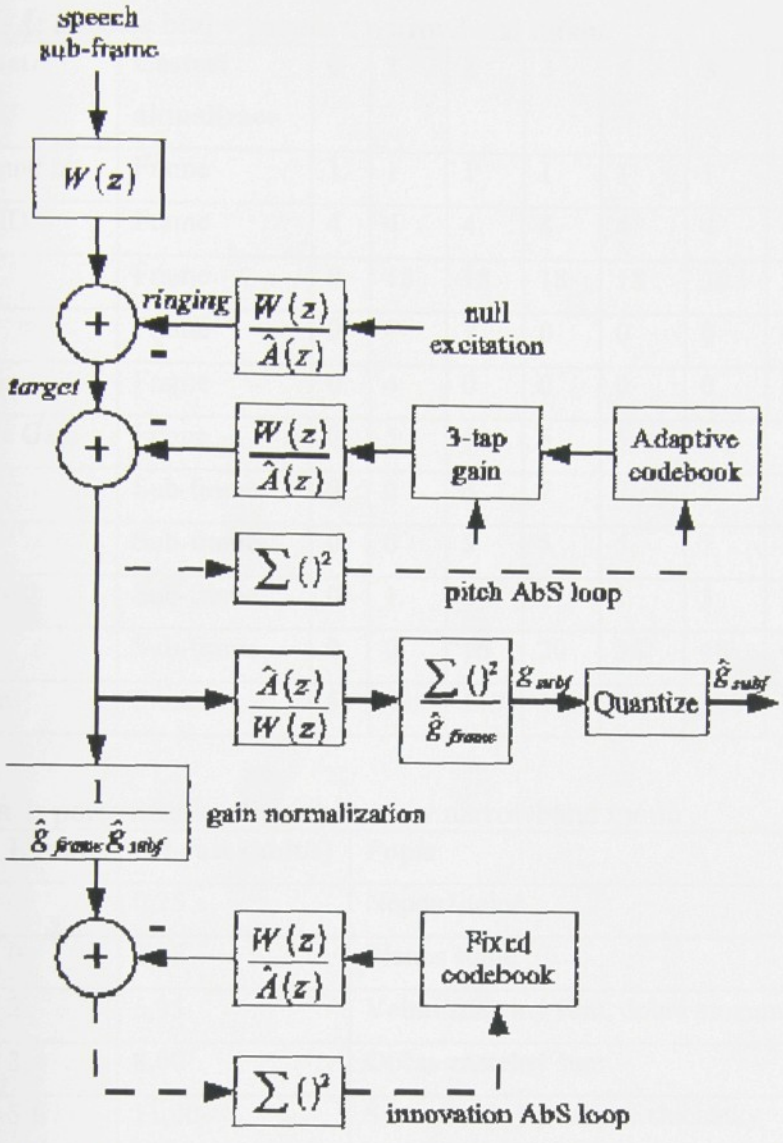
5.3.1 Analýza framů a sub-framů

Analýza pomocí „linear prediction" je provedena pouze jednou na celý frame. Ten je vynásoben asymetrickým Hammingovým okénkem se středem na začátku čtvrtého sub-framu. Výpočet probíhá podle obrázku 4 (koeficienty OL).

Dále jsou prováděny další operace s jednotlivými sub-framy podle obrázku 5 (koeficienty FP, PG, IG a VQ).



Obrázek 4: schéma analýzy celého framu [viz http://speex.org]



Obrázek 5: schéma analýzy sub-framů [viz http://speex.org]

5.3.2 Množství bitů pro jednotlivé parametry v jednom paketu

Speex v narrowband módu umožňuje až 8 různých velikostí datového toku. Od 250 bit/s po 24.6 kbit/s. Zvuk komprimovaný méně než 5.9 kbit/s už není vhodný pro přenos řeči. Váhu parametrů v jednom paketu zobrazuje tabulka 1.

Tabulka 1: alokace bitů v paketu v narrowband módu

Parametr	**Častost aktualizace**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Wideband bit	Frame	1	1	1	1	1	1	1	1	1
Mode ID	Frame	4	4	4	4	4	4	4	4	4
LSP	Frame	0	18	18	18	18	30	30	30	18
OL P	Frame	0	7	7	0	0	0	0	0	7
OL PG	Frame	0	4	0	0	0	0	0	0	4
OL Exc Gain	Frame	0	5	5	5	5	5	5	5	5
FP	Sub-frame	0	0	0	7	7	7	7	7	0
PG	Sub-frame	0	0	5	5	5	7	7	7	0
IG	Sub-frame	0	1	0	1	1	3	3	3	0
VQ	Sub-frame	0	0	16	20	35	48	64	96	10
Celkem	Frame	5	43	119	160	220	300	364	492	79

Tabulka 2: porovnání kvality a bit-rate v narrowband módu

Mód	**Kvalita**	**Bit-rate (kbit/s)**	**Popis**
0	-	0,25	Nepoužitelné
1	0	2,15	Přenos šumu
2	2	5,95	Velmi znatelný šum, dobrá srozumitelnost
3	3-4	8,00	Občas znatelný šum
4	5-6	11,00	Šum znatelný pouze se sluchátky
5	7-8	15,00	S dobrými sluchátky se dá poznat rozdíl
6	9	18,20	Velmi zřídka znatelné rozdíly
7	10	24,60	Absolutně čistý hlas, vhodné i pro muziku
8	1	3,95	Velmi znatelný šum, dobrá srozumitelnost

5.3.1 Kvalita versus datový tok

Tabulka 2 vyjadřuje závislost mezi zvolenou kvalitou, velikostí datového toku a subjektivním posouzením kvality řeči.

5.4 Speex wideband mód

V tomto módu Speex využívá „quadrature mirror filter", aby rozdělil pásmo na dvě pásma o poloviční vzorkovací frekvenci. Tedy 16kHz signál je rozdělen na dva 8kHz signály. Jeden reprezentuje nízké frekvence (0-4kHz), druhý vysoké frekvence (4-8kHz). Spodní pásmo je kódováno stejným způsobem jako signál v narrowband módu (viz kapitola 5.3).

Horní pásmo (4-8kHz) je kódováno následujícím způsobem:

1. Lineární predikce – z větší části probíhá stejně jako v narrow módu. Jediný rozdíl je v množství použitých bitů v paketu. Zatímco v narrow módu zabírá koeficient LSP 18 nebo 30 bitů, ve wideband módu zabírá pouze 12 bitů.
2. Vektorová kvantizace – probíhá stejně jako v narrowband módu.

5.4.1 Množství bitů pro jednotlivé parametry v jednom paketu

Speex ve wideband módu umožňuje 11 různých velikostí datového toku, od 3,95 kbit/s po 42.4 kbit/s. Váhu parametrů pro horní pásmo v jednom paketu zobrazuje tabulka 3, váhu parametrů pro dolní pásmo tabulka 1.

Tabulka 3: alokace bitů v paketu ve wideband módu

Parametr	**Častost aktualizace**	**0**	**1**	**2**	**3**	**4**
Wideband bit	Frame	1	1	1	1	1
Mode ID	Frame	3	3	3	3	3
LSP	Frame	0	12	12	12	12
EG	Sub-frame	0	5	4	4	4
VQ	Sub-frame	0	0	20	40	80
Celkem	Frame	4	36	112	192	352

5.4.2 Kvalita versus datový tok

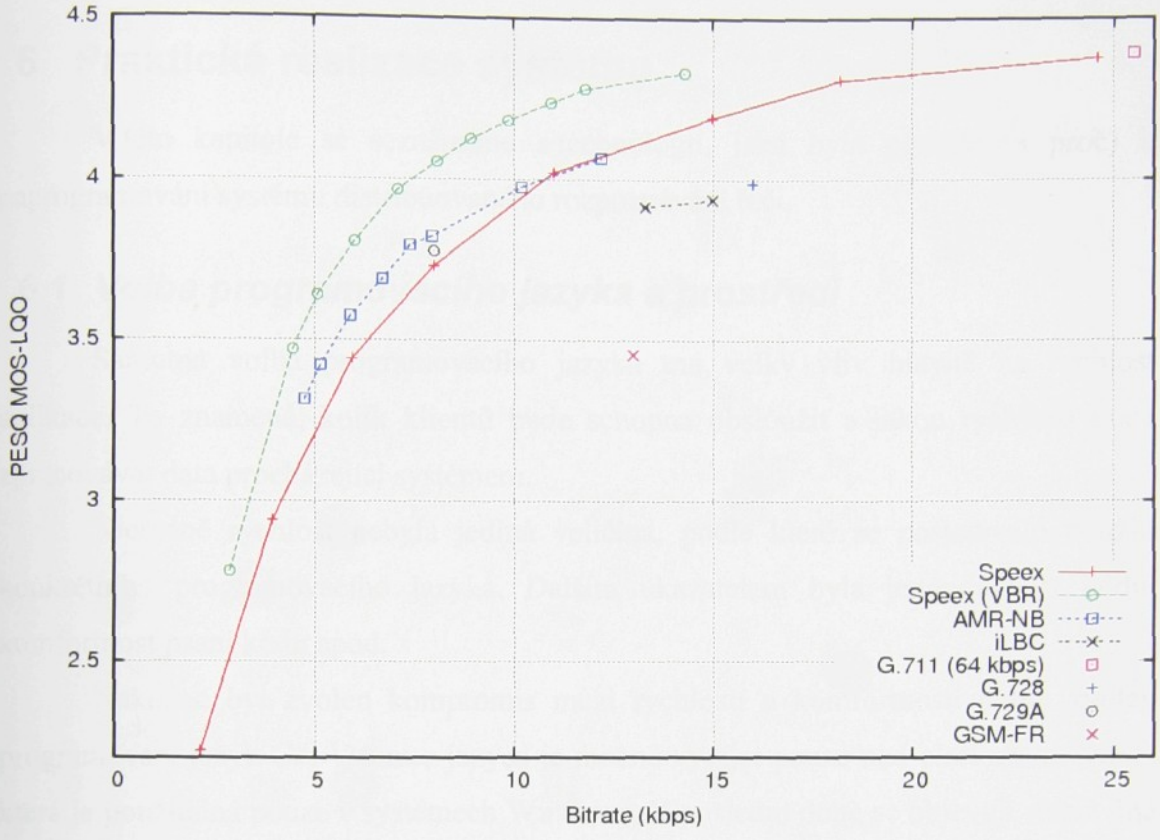
Následující tabulka vyjadřuje závislost mezi zvolenou kvalitou a velikostí datového toku a subjektivním posouzením kvality řeči.

Tabulka 4: porovnání kvality a bit-rate ve wideband módu

Mód/Kvalita	Bit-rate (kbit/s)	Popis
0	3,95	Nesrozumitelné, přenos šumu
1	5,75	Velmi znatelný šum, špatná srozumitelnost
2	7,75	Velmi znatelný šum, dobrá srozumitelnost
3	9,80	Občas znatelný nepříjemný šum
4	12,80	Občas znatelný šum
5	16,80	Zřídka znatelný šum
6	20,60	S kvalitními sluchátky je možné poznat rozdíl
7	23,80	S kvalitními sluchátky je možné poznat rozdíl
8	27,80	Ani s kvalitními sluchátky není znatelný rozdíl
9	34,40	Ani s kvalitními sluchátky není znatelný rozdíl
10	42,40	Skvělé na kódování řeči, vhodné i pro hudbu

5.5 Speex v porovnání s ostatními VoIP kodeky

Vzhledem k jeho úzké specializaci a relativně moderním technologiím Speex vyniká nad ostatními kodeky, ale není příliš hojně využívaný z důvodů konzervativnosti firem vyrábějící technologie pro VoIP. Na grafu 1 a tabulce 5 je možné vidět porovnání kodeku Speex s ostatními kodeky požívanými v oblasti internetové telefonie. Za povšimnutí stojí hlavně jeho univerzálnost a nízké datové toky při zachování srozumitelnosti zvukového signálu.



Graf 1: porovnání speex a ostatních kodeků [viz http://speex.org]

Tabulka 5: porovnání nejznámějších kodeků používaných pro kompresi lidské řeči.

Kodek	Frekv. (kHz)	Bit-rate (kbit/s)	Zpoždění (ms)	Din. změna bit-rate	VBR	Ochrana proti rušení signálu
Speex	8,16,32	2,15-24,6 (NB) 4-44,2 (WB)	30 (NB) 34 (WB)	Ano	Ano	Ne
iLBC	8	15,2 nebo 13,3	25 nebo 40	Ne	Ne	Ne
AMR-NB	8	4,75-12,2	25	Ano	Ne	Ano
G.722.2	16	6,6-23,85	25	Ano	Ne	Ano
G.722.1	16	16, 24 nebo 32	40	Ano	Ne	Ano
G.729	8	8	15	Ne	Ne	Ano
GSM-FR	8	13	20	Ne	Ne	-
GSM-EFR	8	12,2	20	Ne	Ne	Ano
G.723.1	8	5,3 nebo 6,3	37,5	Ne	Ne	-
G.728	8	16	0,63	Ne	Ne	Ne
G.722	16	48, 56 nebo 64	-	Ne	Ne	-

6 Praktická realizace systému

V této kapitole se seznámíme s technologií, jaká byla použita (a proč) k naprogramování systému distribuovaného rozpoznávání řeči.

6.1 Volba programovacího jazyka a prostředí

Samotná volba programovacího jazyka má velký vliv hlavně na rychlost aplikace. To znamená, kolik klientů bude schopna obsloužit a jakou rychlostí bude zpracovávat data procházející systémem.

Nicméně rychlost nebyla jediná veličina, podle které se posuzovalo použití konkrétního programovacího jazyka. Dalším ukazatelem byla jednoduchost kódu, komfortnost psaní kódu apod.

Nakonec byl zvolen kompromis mezi rychlostí a komfortností a byl zvolen programovací jazyk C#. V tomto jazyce je možné vyvíjet pouze nad platformou .NET, která je použitelná pouze v systémech Windows. V poslední době se objevují různé jiné freeware interprety tohoto jazyka, které je možné použít i v systémech Linux nebo Mac. Jedním z nich je například Mono (viz http://www.mono-project.com).

Po zvolení programovacího jazyka byla volba prostředí téměř jasná. Vzhledem k jednoduchosti, množství nástrojů a pomůcek zvítězilo Visual Studio 2008. Jinou alternativou bylo freeware vývojové prostředí MonoDevelop, které je vyvíjeno v rámci projektu Mono.

6.2 Volba síťové technologie

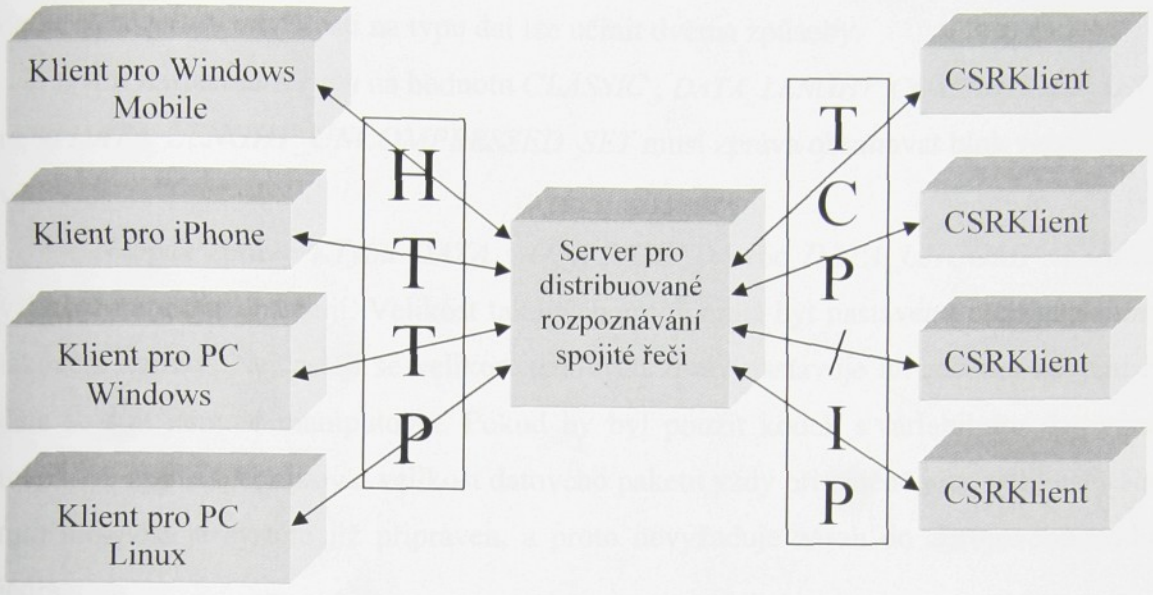
V tomto ohledu byla volba snadná. Vzhledem k požadavkům na rychlost a extrémní paketovatelnost (jsou přenášeny malé pakety v krátkých časových intervalech) bylo nutné použít protokol s co možná nejnižší redundancí a režií přenosu.

Proto je pro komunikaci mezi serverem a CSRKlientem použit protokol TCP/IP bez jakékoliv nadstavby a pro komunikaci s GUIKlientem je použit protokol HTTP.

Protokol HTTP byl použit z důvodu požadavku na průchodnost komunikace skrze webové proxy servery a firewally. Tato schopnost byla úspěšně testována pouze v „laboratorních" podmínkách a ne ve skutečné síťové infrastruktuře.

Protokol TCP byl zvolen hlavně z důvodu zaručeného doručení paketu, což protokol UDP neumožňuje. V telefonních hovorech je hlas přenášen pomocí protokolu

UDP, neboť občasné výpadky nemají pro člověka na srozumitelnost řeči téměř žádný vliv. To neplatí pro technologii v2t, která je na výpadky částí slov velmi citlivá.



Ilustrace 1: základní topologie systému distribuovaného rozpoznávání řeči

6.2.1 HTTP chunked

Vzhledem ke stavové povaze protokolu HTTP bylo nutné nějakým způsobem obejít tuto vlastnost, protože navazování spojení pro každý paket by bylo velmi pomalé. Z toho důvodu bylo použito chunked kódování těla paketu HTTP. To znamená, že je odeslána hlavička, kde není specifikována velikost těla a samotné tělo je přenášeno „po částech". Každá část je odeslána samostatně a spojení je stále otevřené. Tento princip se využívá u webových prohlížečů ke stahování nebo nahrávání dat na server. Konec takto přenášené části těla není nikterak specifikován, proto server není schopen rozpoznat, zda data, která přišla, patří ještě do předchozího paketu nebo představují již nový paket. Bohužel stejné chování má i protokol TCP, a tak bylo nutné navrhnout systém zpráv, ve kterých by byla specifikována velikost a typ přenášených dat (viz kapitola 6.3.1).

6.3 Výsledná realizace komunikace

Tato kapitola detailně popisuje způsoby komunikace mezi částmi systému. Popisuje různé typy zpráv a jejich význam, dále vnitřní a komunikační logiku systému a jeho komponent v souvislosti s navazováním, ukončováním nebo průběhem komunikace.

6.3.1 Zprávy systému

Jak již bylo zmíněno v kapitole 6.2.1, není možné při přenosu paketů určit velikost přenášených dat. Z toho důvodu je nutné dávat protistraně vědět, jak dlouhá data přicházejí. V závislosti na typu dat lze učinit dvěma způsoby.

Při nastavení *KTypu* na hodnotu *CLASSIC* , *DATA_LENGHT_COMPRESSED_SET* nebo *DATA_LENGHT_UNCOMPRESSED_SET* musí zpráva obsahovat blok *velikost* vždy.

Naopak zprávy *KTypu DATA_COMPRESSED* nebo *DATA_UNCOMPRESSED* velikost v sobě neobsahují. Velikost takových zpráv musí být nastavena před posláním takovéto zprávy. Nejčastěji se velikost datových zpráv nastavuje na začátku spojení a dále se s ní nemusí manipulovat. Pokud by byl použit kodek s variabilním datovým tokem, je nutné specifikovat velikost datového paketu vždy při změně jeho velikosti. Na tuto možnost je systém již připraven, a proto nevyžaduje zásah do zdrojového kódu jádra.

- Klasická zpráva:

  V případě, že je *Velikost* rovná nule, vynechává se blok *Data*.

  Tabulka 6: rozložení dat v klasické zprávě

KTyp	Typ	Velikost	Data

- Zpráva nastavující velikost zvukových dat:

  Tabulka 7: rozložení dat ve zprávě nastavující velikost zvokových dat

KTyp	Velikost

- Zpráva obsahující zvuková data:

  Tabulka 8: rozložení dat ve zprávě se zvukovými daty

KTyp	Data

a) KTyp

Informace o velikosti jednoho bytu reprezentující typ přenášené zprávy, kterému rozumí pouze systém distribuovaného rozpoznávání řeči, nikoliv samotný rozpoznávač.

Hodnoty, které může nabývat, reprezentuje výčtový typ *EKMessageTypes*. Jejich seznam a stručný popis obsahuje kapitola 8.3.1.

b) Typ

Tento blok paketu je ve většině zpráv posílaných GUIKlientem vynecháván, protože GUIKlient odesílá převážně zvuková data. V podstatě jediné pakety, které se přenášejí, na navázání spojení a spuštění rozpoznávače obsahují typ zprávy.

Zprávy přicházející do GUIKlienta naopak obsahují tento typ vždy. Vyjadřuje, zda se jedná o rozpoznaný textový řetězec, zpoždění rozpoznávání, chybové hlášení atd.

Těmto údajů rozumí, na rozdíl od *KTypu*, pouze rozpoznávač. Systém distribuovaného rozpoznávání řeči jim rozumět nemusí. Výjimkou je typ *OPERATIVE*, který slouží pro signalizaci v systému.

Typ je v aplikacích reprezentován výčtovým typem *EMessageTypes*, popsaným v kapitole 8.3.1.

c) Velikost

Reprezentuje velikost následujících dat. V přenosu směrem od GUIklienta k CSRklientovy se téměř nepoužívá, protože velikost zvukových dat je určena jiným způsobem (viz kapitola 6.3.1).

d) Data

Libovolně dlouhý blok dat. Jeho velikost je specifikována v předchozím bloku, s výjimkou zvukových dat. Může obsahovat jakoukoliv informaci, jejíž typ je specifikován blokem *typ* nebo *ktyp*.

6.3.2 Navazování spojení

V následujících kapitolách je popsán princip navazování spojení mezi jednotlivými aplikacemi (GUIKlient, Server, CSRKlient).

a) GUIKlient – Server

Navazování spojení mezi GUIKlientem a serverem probíhá ve třech fázích:

1. Klient odešle svoje údaje (uživatelské jméno, ...) a jako odpověď dostává buď zamítnutí spojení (v případě, že není volný žádný rozpoznávač) nebo potvrzení spojení a číslo session. Dále se klient hlásí už jenom pomocí tohoto čísla.
2. Klient naváže spojení pro zápis.
3. Klient naváže spojení pro čtení.

Signalizace v těchto případech probíhá za pomoci proměnných přenášených v hlavičce HTTP dotazu nebo jako proměnných v URL dotazu.

Hlavička HTTP musí obsahovat vždy proměnnou „user-agent" nastavenou na hodnotu „kerberos_klient". Dále musí obsahovat položku „user-name", kde je uloženo jméno klienta připojujícího se k systému.

- První připojení:
  - Dotaz na adresu:

http://[adresa_serveru]/ProxyServerlistener?connectionType=connect

  - Odpověď: Textový řetězec s informací, zda je volný CSRKlient a případné číslo session. Příklady odpovědí:

1. <HTML><BODY>NO_IDLE_CSR</BODY></HTML>
   - V tomto případě není volný žádný rozpoznávač a klient musí počkat.
2. <HTML><HEAD><META>2</META></HEAD><BODY>COUPLED</BODY></HTML>
   - Spojení je možné navázat na session číslo 2.

- Připojení Writeru:
  - Dotaz na upload na adresu:

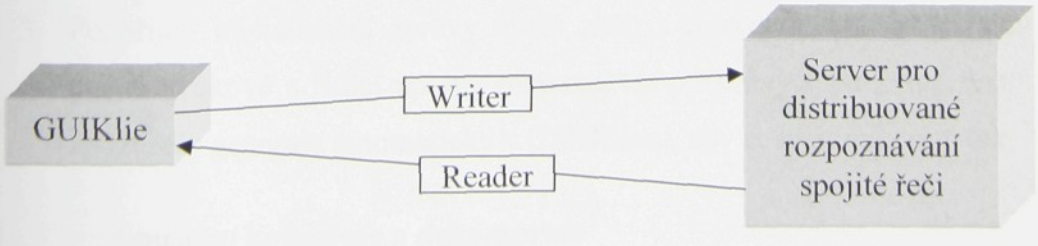
http://[adresa_serveru]/ProxyServerlistener?connectionType=writer&sessionID=2

  - Odpověď není. Spojení se drží po dobu diktování.

- Připojení Readeru:
  - Dotaz na download z adresy:

http://[adresa_serveru]/ProxyServerlistener?connectionType=reader&sessionID=2

  - Odpověď: Textový řetězec „welcome reader".



Ilustrace 2: připojený GUI klient k serveru

b) CSRKlient – Server

Navazování komunikace je v tomto případě velmi jednoduché. Klient se pouze pokusí připojit k zadané IP adrese a portu. V případě, že server na této adrese a portu poslouchá, je klient spojen.

Po spojení je na serveru vygenerováno nové číslo session, které reprezentuje tento nově připojený CSRKlient. Pokud bude spárován s nějakým GUIKlientem, pak GUIKlient obdrží toto číslo session. Číslo session proto není v průběhu času unikátní (po odpojení jednoho GUIKlienta muže stejné číslo získat jiný klient), avšak je unikátní v rámci běžících rozpoznávání a tudíž i připojeného GUIKlienta. Rozdělování čísel session znázorňuje ilustrace 3.

6.3.3 Začátek rozpoznávání

Pokud se připojí GUIKlient, je spárován s volným CSRKlientem. Od této chvíle se mohou posílat libovolné zprávy, kterým nemusí server rozumět. Struktura zprávy musí zůstat stejná. Před samotným spuštěním je nutné nastartovat procesy, které se starají o kompresi a dekompresi zvukových dat.

1. GUIKlient odešle parametry příkazové řádky, se kterými je rozpoznávač spuštěn. Po spuštění rozpoznávač odesílá textovou zprávu oznamující spuštění rozpoznávání. Parametry příkazové řádky jsou posílány jako textová s ktypem CLASSIC a typem OPERATIVE, kde jsou parametry řádky uvozeny textem „cmd: ".
2. Dále musí GUIKlient specifikovat, jak velké budou pakety se zvukovými daty. To může kdykoliv během přenosu změnit. V závislosti na preferovaném způsobu přenosu, komprimovaném nebo nekomprimovaném, zvolí KTyp zpráv, jak pro nastavení velikosti dat, tak pro samotný přenos dat. Nastavení velikosti zvukových paketů je popisováno v kapitole 6.3.1.
3. Po přijetí inicializační zprávy může začít s přenosem dat. GUIKlient odesílá pouze zvukové a řídící zprávy rozpoznávače. Zprávy typu „vrať text" a „vrať zpoždění" generuje automaticky v CSRKlient, aby se ušetřil datový tok.

- **Spuštění komprese a dekomprese**

Oba procesy (kompresní a dekompresní) jsou externí spustitelné soubory využívané třídou CodecImpl (viz kapitola 8.4.1), která s nimi komunikuje veskrze

standardní vstupy a výstupy. Tyto externí soubory se starají o kompresi a dekompresi zvukových dat pomocí kodeku Speex.

Třídu CodecImpl je možné nahradit vlastní implementací a tu poté k aplikaci připojit. Postup je popsán v kapitole 6.4 a umožňuje rozšiřovat možnosti komunikace i o šifrování a jinou funkčnost.

6.3.4 Ukončení rozpoznávání

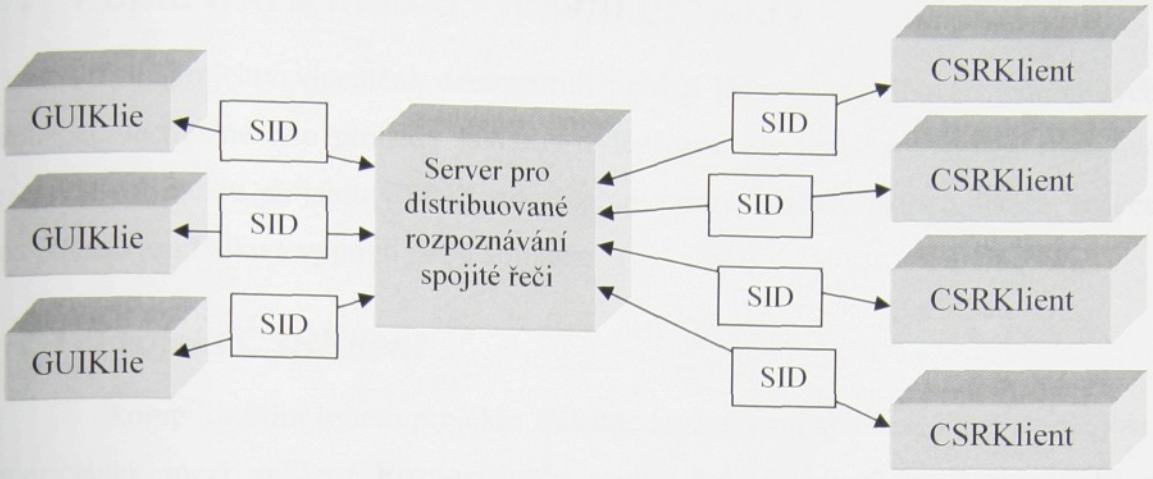
Rozpoznávač je možné ukončit klasickým způsobem, tj. odesláním zprávy „konec session". Ale tomu server „nerozumí". Proto je potřeba odeslat speciální signalizační zprávu serveru, že rozpoznávání končí (signalizační typy zpráv popisuje kapitola 8.3.1). V takovém případě server odpojí GUIKlienta a ukončí párování s příslušným CSRKlientem. CSRKlient na tuto zprávu reaguje ukončením rozpoznávače a návratu do stavu, kdy čeká na parametry příkazové řádky pro další spuštění rozpoznávače.

6.3.5 Průběh rozpoznávání

Samotné rozpoznávání tak probíhá téměř bez účasti Serveru a CSRKlienta. Ti pouze předávají zprávy mezi GUIKlientem a rozpoznávačem. Obě aplikace pouze převádějí zprávy z jednoho rozhraní na druhé a naopak.

Server se stará, aby zprávy chodily vždy od konkrétního GUIKlienta ke konkrétnímu CSRKlientovi. Má na starosti hlídání obou stran, zda odpovídají v časovém limitu (jestli se neodpojili nebo nedošlo k jejich pádu). V případě timeoutu jedné ze stran, je odeslána ukončovací zpráva protistraně a spojení je ukončeno (viz kapitola 6.3.4).

CSRKlient během rozpoznávání přenáší zprávy mezi rozpoznávačem, který má spuštěný jako podproces, a serverem. Generuje zprávy „vrať text" a „vrať zpoždění". V případě komprimovaného přenosu se stará o dekompresi zvukových dat.



Ilustrace 3: systém rozdělování čísel session (sid)

6.4 Připojení vlastního kódovacího pluginu

Systém umožňuje připojení vlastní implementace pluginu využívaného ke kódování zvukových dat. V současné době existuje pouze jedna implementace pluginu, která využívá ke kódování kompresní kodek Speex, ale nedovoluje šifrování ani jiné úpravy přenášených dat.

Vlastní implementaci pluginu je velice jednoduché k systému připojit. Třída implementující rozhraní ICodec (viz kapitola 8.4.1) musí být umístěna do knihovny codec.dll. Tato knihovní třída se poté nahraje do adresáře s GUIKlientem a CSRKlientem. Systém již automaticky rozpozná knihovnu a bude ji využívat ke kódování a dekódování zvukových dat.

Takto je možné k systému připojit například i šifrovací algoritmus a přenos zvukových dat šifrovat. Vedlejším efektem ale bude nemožnost využívat nahraná zvuková data na serveru. Textovou komunikaci nelze v současné době šifrovat.

7 Popis tříd a metod - hlavní projekty

Tyto projekty víceméně demonstrují použití jádrových knihoven jednotlivých aplikací. Je možné tyto projekty jakkoliv vylepšovat, ale funkčnost systému zůstává stejná. S výjimkou projektu CSRKlient obsahují pouze spuštění jádra aplikace, reakce na příkazovou řádku a vypnutí jádra aplikace.

7.1 Projekt CSRKlient

Zkompilováním tohoto projektu získáme konzolovou aplikaci, která slouží jako mezičlánek mezi aplikací Rozpoznávače spojité řeči (vyvíjené na ústavu ITE) a serverem distribuovaného rozpoznávání řeči. Tento klient se stará o připojení k serveru, spuštění rozpoznávače, přenos zpráv mezi serverem a rozpoznávačem a zasílání speciálních zpráv rozpoznávači.

7.1.1 Jmenný prostor CSRKlient

Pod tímto jmenným prostorem se nachází veškeré třídy projektu CSRKlient.

- Třída CSRKlient

Tato třída má mimo jiné na starosti spuštění a komunikaci s rozpoznávačem spojité řeči. Dále komunikaci se serverem a zasílání zpráv „vrať text" a „vrať zpoždění" s určenou periodou.

Zprávy přijaté od serveru odesílá na standardní vstup rozpoznávače a ze standardního výstupu čte rozpoznaný text, který odesílá na server.

- Třída StartupSession

Přepravka udržující informace o právě spuštěné session.

- Třída WaitForSession

Tato třída má na starosti sběr dat potřebných pro spuštění rozpoznávače a započetí jeho práce. Tyto data přicházejí od serveru. Ten je přeposílá od GUIKlienta.

- Třída Program

Zajišťuje spuštění aplikace a stará se o provedení příkazů zadávaných uživatelem přes příkazovou řádku.

7.2 Projekt GUIKlientLite

Tento projekt reprezentuje testovací aplikaci, která byla využívána během vývoje. V jejím zdrojovém kódu je demonstrováno použití knihovny GUIKlientCore (viz kapitola 8.2). Aplikace se dokáže spojit se serverem distribuovaného rozpoznávání řeči. Je schopna nahrávání zvuku a jeho zasílání na server. Ze serveru čte vrácený text, který bez jakéhokoliv zpracování zobrazuje na obrazovku.

7.2.1 Jmenný prostor GUIKlientLite

Obsahuje třídu MainWindow, grafické rozhraní aplikace.

- Třída MainWindow

Třída MainWindow reprezentuje hlavní grafické okno aplikace. Na tomto okně jsou k dispozici ovládací a zobrazovací prvky.

- Třída Program

Tato třída se stará o spuštění aplikace.

7.2.2 Jmenný prostor RecordingJZ

V tomto jmenném prostoru jsou obsaženy třídy zajišťující nahrávání zvuku za pomoci DirectShow. Původním autorem tříd je ing. Jindřich Žďánský, Ph.D.

- Třída JZRecorder

Tato třída nahrává zvuk z primárního zdroje pomocí DirectShow. Nahraný zvuk se předává v bufferu pomocí události nesoucí třídu JZEventArgs. Nevýhodou této metody je nemožnost nastavit konkrétní velikost bufferu pro zvuková data. DirectShow může vracet data i po relativně velkých blocích, což způsobuje zpoždění rozpoznávání.

- Třída JZEventArgs

Argumenty události JZRecorderu. Mezi argumenty stojí za zmínku buffer nesoucí nahraná zvuková data.

7.3 Projekt DistributionServer

V tomto projektu je demonstrováno použití knihovny DistributionServerCore (viz kapitola 8.1).

7.3.1 Jmenný prostor DistributionServer

Tento jmenný prostor obsahuje pouze třídu Program, starající se o spuštění serveru distribuovaného rozpoznávání řeči.

- Třída Program

Tato třída se stará o spuštění serveru distribuovaného rozpoznávání řeči. Dále reakcemi na příkazy z příkazové řádky zadané uživatelem a ukončení serveru.

8 Popis tříd a metod – knihovny

Tyto knihovny obsahují převážně důležité třídy pro běh celého systému. Jejich upravováním je možné upravovat charakteristiky a chování celého systému distribuovaného rozpoznávání řeči.

8.1 Projekt DistributionServerCore

Zkráceně DSC, je stěžejní projekt celého systému. Reprezentuje jádro serveru distribuovaného rozpoznávače řeči. Toto jádro se stará o navazování a ukončování spojení s klienty, přenos dat mezi klienty a ukládání komunikace na pevný disk. Dále se stará o kontrolu připojení s klienty a odpojování neaktivních klientů.

8.1.1 Jmenný prostor DSC.Distribution.Server.Clients

Tento jmenný prostor obsahuje třídy reprezentující fyzické klienty. Tyto třídy obsahují rozhraní pro komunikaci s fyzickými klienty.

- Třída HttpClient

Reprezentuje GUIklienta připojeného přes protokol HTTP.

- Třída WinsockClient

Reprezentuje CSRKlienta připojeného pomocí protokolu TCP/IP.

8.1.2 Jmenný prostor DSC.Distribution.Server.Communicators

Zde se nachází třída zajišťující komunikaci mezi párem rozhraní klientů.

- Třída KQueue

Tato třída tvoří most mezi HttpClientem a WinsockClientem. Dále zajišťuje synchronizaci dat. Oba klienti obsahují referenci na instanci této třídy a prostřednictvím ní si předávají zprávy. Třída je synchronizovaná, jelikožkaždý klient běží v jiném vlákně.

8.1.3 Jmenný prostor DSC.Distribution.Server.Core

Jmenný prostor obsahující třídy zajišťující běh samotného serveru.

- Třída HttpClientListener

Má na starosti registraci GUIklientů, kteří žádají o připojení. Připojení probíhá přes protokol HTTP. Samotné navázání spojení je delegováno na implementaci rozhraní IClientManager (v tomto případě se jedná o třídu DistributionServerCore).

Připojení klienta probíhá ve třech fázích. Nejprve se klient prokáže uživatelským jménem a požádá o připojení. Jako odpověď je mu sděleno, zda je volný CSRKlient a pokud ano, tak i číslo session. Dále se klient prokazuje pod tímto číslem. V druhé fázi se klient připojí s požadavkem na upload dat a obdrží stream pro posílání zpráv. V další fázi se připojí s požadavkem na download dat a získá stream pro čtení zpráv. Po absolvování třetí fáze je klient spojen s CSRklientem a mohou komunikovat.

- Třída TcpClientListener

Tato třída má na starosti registraci Winsockklientů, kteří žádají o připojení. Připojení probíhá přes protokol TCP/IP a samotné navázání spojení je delegováno na implementaci rozhraní IClientManager, stejně jako v případě HttpKlientů.

- Třída DistributionServerCore

Hlavní třída jádra serveru distribuovaného rozpoznávání řeči, která má na starosti spuštění, běh a ukončení tohoto serveru. Mimo jiné se také stará o navazování spojení s klienty, hlídání timeoutů, ukončování spojení a párování klientů.

- Interface IClientManager

Třída implementující toto rozhraní se musí starat o navazování a ukončování spojení s klienty, jejich evidenci a párování. Jedinou implementací je v současné době DistributionServerCore.

8.2 Projekt GUIKlientCore

Projekt obsahuje zdrojové kódy jádra uživatelského klienta serveru. Jádro GUI klienta se stará o navázání, držení a ukončení spojení. Obsahuje třídy pro komunikaci se serverem. O událostech, včetně nově příchozích dat, je spouštějící instance informována využitím událostí. Použití této knihovny je demonstrováno v projektu GUIKlient (viz kapitola 7.2).

8.2.1 Jmenný prostor GUIKlientCore.comManagement

Jsou zde obsaženy třídy jádra uživatelského klienta starající se převážně o zprávu spojení a přenos dat.

- Třída HttpReader

Třída je zodpovědná za čtení dat (v našem případě textu) přicházejících ze serveru distribuovaného rozpoznávání řeči, ke kterému je klient připojen. Data jsou převedena na řetězec a je vyvolána událost nově příchozích dat.

- Třída HttpWriter

Třída zajišťující odeslání zprávy na server.

- Třída HttpClient

Hlavní součást jádra GUIklienta. Obstarává navázání spojení a spuštění instance třídy HttpReader. O jakýchkoliv změnách ve spojení se serverem informuje pomocí událostí.

8.3 Projekt CommonTools

V tomto projektu se nacházejí převážně třídy společné všem částem systému distribuovaného rozpoznávání řeči. Mimo jiné obsahuje i univerzální třídu na čtení a zápis zpráv do streamu.

8.3.1 Jmenný prostor Common.Tools

Jmenný prostor projektu CommonTools.

- Třída KAsyncStreamReader

Třída zajišťující sekvenční čtení celistvých zpráv ze zadaného datového proudu. Zpráva musí být ve tvaru *KType->Data* nebo *KType->Type->Size->Data*. Po přečtení celé zprávy je vyvolána událost „nová data". Čtení zpráv běží v samostatném vlákně a neblokuje zbytek aplikace.

- Třída KAsyncStreamWriter

Zapisuje celistvou zprávu zadaného datového proudu. Je možné nastavení synchronní/asynchronní zápis.

- Třída KAsyncStreamReaderEventArgs

Přepravka pro nově přijatou zprávu. Využívá se jako parametr události „nová data" třídy KAsyncStreamReader.

- Třída NanocoreMessage

Přepravka pro jednu zprávu. Obsahuje *KTyp*, *Typ*, *Velikost* a *Data*.

- Třída ShowMessage

Tato třída se stará o zobrazení zprávy o chybě nebo upozornění správci systému.

- Třída OperativeInfoFactory

Vytváří režijní zprávy.

- Třída CodecEventArgs

Přepravka pro předávání dat v událostech kodeku. Kodek je implementace rozhraní ICodec, která je schopná zakódovat a dekódovat předaná zvuková data.

- Interface ICodec

Třída implementující toto rozhraní je zodpovědná za kódování (kompresi) a dekódování (dekompresi) zvukových dat přenášených systémem.

- Výčtový typ EClientTypes

Výčtový typ jednotlivých klientů. V současné době existují pouze dva klienti, Http a Tcp, ale do budoucna je možné přidávat další (např. VoIP klient).

- Výčtový typ EKMessageTypes

Seznam Ktypů zpráv:

- CLASSIC – Zpráva klasického typu. Bude následovat Typ zprávy, velikost a data.
- DATA_LENGHT_COMPRESSED_SET – Zpráva nastavující velikost nekomprimovaného datového bloku. Zpráva dále obsahuje Typ zprávy (z důvodu zachování kompatibility), který může být nulový a velikost datového bloku.
- DATA_LENGHT_UNCOMPRESSED_SET – Stejný druh zprávy jako v předchozím případě, pouze udává velikost nekomprimovaných dat.

- DATA_COMPRESSED – Zpráva obsahuje pouze komprimovaná data o dříve specifikované velikosti.
- DATA_UNCOMPRESSED – Zpráva obsahuje nekomprimovaná data. Velikost dat musí být specifikována nejdříve zprávou *DATA_LENGHT_UNCOMPRESSED_SET*.

  - Výčtový typ EMessageTypes

Seznam Typů zpráv. Z důvodů ochrany autorských práv není možné zveřejnit seznam typů zpráv.

  - Výčtový typ EOperativeInfos

Seznam režijních zpráv systému. Režijní informace se přenášejí ve zprávách *ktypu CLASSIC* z důvodu kompatibility. Bylo by možné stanovit speciální *ktyp* a formát režijních zpráv a tím šetřit datový tok. V současné době existují tyto režijní zprávy:

- TIMEOUT – Klient byl odpojen v důsledku delší neaktivity a klient na druhé straně spojení je o tom takto informován.
- READY – Klient dává najevo, že je stále funkční. Tuto zprávu posílají CSR klienti, aby server mohl kontrolovat jejich připravenost v době, kdy právě nerozpoznávají.
- EXITING – Klient ukončuje regulérně svoji činnost (např.: Uživatel zavírá GUIKlienta). Druhý klient dostává tuto zprávu, aby se převedl do startovního stavu a čekal na další session.
- UNKNOWN – Neznámá zpráva. Tato zpráva se systémem nepřenáší. Slouží k testování validity připravované zprávy.

  - Výčtový typ EShowMessageType

Výčtový typ reprezentující naléhavost zobrazované zprávy. Využití v třídě ShowMessage. Následuje seznam:

- DEBUG – Ladící informace
- INFO – Základní informace systému
- WARN – Varování
- ERROR – Došlo k chybě
- FATAL – Došlo k fatální chybě

8.3.2 Jmenný prostor Common.Tools.Statistics

Obsahuje třídy starající se o vedení a zobrazení statistik chodu systému. Statistiky se vyvolávají napsáním slova „stat" na konzoly aplikace.

- Třída CSRStatisticsManagerImpl

Třída spravující statistiky aplikace CSR klienta.

- Třída ServerStatisticsManagerImpl

Třída spravující statistiky serveru distribuovaného rozpoznávání řeči.

- Třída StatisticsManagerFactory

Tovární třída vytvářející instance implementací IStatisticManager.

- Interface IStatisticManager

Třída implementující toto rozhraní musí umět spravovat statistiky a zobrazit je na konzoly.

- Výčtový typ EStatisticsManagers

Výčtový typ obsahující všechny dostupné manažery statistik (Server, CSRKlient).

8.4 Projekt Codec

Tento projekt obsahuje zdrojové kódy jednoduché implementace rozhraní ICodec. V našem případě je použit kodek Speex (viz kapitola 5) pro kompresi a dekompresi zvukových dat.

8.4.1 Jmenný prostor Codec

Obsahuje jedinou třídu CodecImpl.

- Třída CodecImpl

Tato třída je implementací rozhraní ICodec. Její primární úkol je spustit encoder nebo dekodér kodeku Speex a přes standardní vstupy a výstupy posílat a číst zvuková data. Encoder data přijímá v surové podobě a vrací zkomprimovaná. Decoder je na druhé straně systému distribuovaného rozpoznávání řeči zase dekomprimuje.

9 Závěr

Výsledkem diplomové práce je funkční systém distribuovaného rozpoznávání řeči, který umožňuje automaticky převádět mluvené slovo na psanou formu. Systém se skládá z grafického (uživatelského) klienta, který je schopen se připojit a komunikovat se Serverem distribuovaného rozpoznávání pomocí protokolu využívajícího HTTP. GUI klient je pouze demonstrativní pro účely testování. Dále systém obsahuje druhého klienta, jehož součástí je Rozpoznávač spojité řeči, vyvíjený laboratoří SpeechLab. Tento klient je také schopen se připojit k Serveru distribuovaného rozpoznávání a komunikovat s ním pomocí vlastního protokolu navrženého nad TCP/IP.

Již zmíněný server poté představuje most mezi těmito klienty. Podporuje připojení většího počtu obou druhů klientů a zprostředkování jejich komunikace. Server má zdokumentované komunikační rozhraní, a proto je možné naprogramovat libovolného GUI klienta komunikujícího s tímto serverem. Veškerá komunikace se na serveru nahrává pro pozdější výzkumné účely. Server provádí správu klientů, kontrolu začátku a konce komunikace a odpojení klienta při delší nečinnosti.

Celý systém je naprogramován v programovacím jazyce C#, který byl zvolen z důvodu rychlého vývoje aplikací a komfortu programování. Výjimkou je kompresní plugin využívající kodek Speex, který je naprogramován v jazyce C++. Z výše zmíněných důvodů je nutné mít na počítači nainstalovaný .NET Framework ve verzi nejméně 3.0.

Pro komunikaci mezi uživatelským klientem a serverem byl zvolen protokol HTTP s kódováním chunked. HTTP Protokol byl zvolen kvůli jeho bezproblémové prostupnosti skrze internetové proxy servery. CSR klient se serverem komunikuje vlastním protokolem, který byl navržen s ohledem na rychlost. Oba protokoly fungují nad TCP/IP, které bylo zvoleno kvůli garantování doručení paketu a nehrozí tudíž ztráta žádných zvukových dat.

Pro přenos zvukových dat od uživatelského klienta k CSR klientovi je možné využít kodek Speex pro kompresi a dekompresi. Takto bylo možné snížit datový tok z cca 320kbit/s na 20kbit/s. Problém nastává při použití v mobilních zařízeních, pro které v současné době neexistuje implementace kodeku Speex a je nutné data přenášet bez použití komprese. Proto systém umožňuje data přenášet komprimovaně i nekoprimovaně.

Systém splňuje všechny požadavky zadání, včetně požadavků vzniklých během vývoje. Byl testován v laboratorních podmínkách pro dva uživatelské a dva CSR klienty a vykazoval velmi dobrou stabilitu. Je schopen si poradit i s výpadkem jednoho z klientů a zachová se definovaným způsobem. Je možné systém dále vylepšovat nebo modifikovat jeho stávající funkčnost.

Za zmínku stojí možnost připojit server na VoIP ústřednu a umožnit tak uživateli diktovat přímo do telefonu bez nutnosti dalších zařízení a aplikací. To by bylo výhodné například při konferenčních hovorech apod. Některé VoIP telefony umožňují zobrazit text na displeji, aby uživatel mohl kontrolovat textovou podobu mluveného slova.

Dále je možné k systému připojit šifrovací plugin namísto kompresního pluginu a docílit tak šifrované komunikace. Bohužel nebylo dříve myšleno na možnost šifrování textových zpráv, a proto je tato možnost bez zásahu do zdrojových kódů nedostupná.

Použitá literatura

[1] C# [online] www: http://www.csharp-home.com

[2] TROELSEN, Andrew. *C# a .NET 2.0 profesionálně.* Zoner, 2006. ISBN: 80-86815-42-0

[3] MSDN [online] www: http://msdn.microsoft.com

[4] VoIP [online] www: http://www.iec.org/online/tutorials/int_tele/index.asp

[5] SIP [online] www: http://www.ietf.org/rfc/rfc3261.txt

[6] HTTP [online] www: http://www.w3.org/Protocols/rfc2616/rfc2616.html

[7] URI [online] www: http://tools.ietf.org/html/rfc3986

[8] WinSock 2 [online] www: http://msdn.microsoft.com/en-us/library/ms740673.aspx

[9] Laboratoř Speechlab [online] www: http://www.ite.tul.cz/speechlab/index.php