MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Vstupní modul pro VoiceXML interpretr JVoiceXML

BAKALÁŘSKÁ PRÁCE

Matúš Kempa

Brno, jaro 2009

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Vedúci práce: Mgr. Luděk Bártek, PhD.

Poďakovanie

Touto cestou by som rád poďakoval vedúcemu svojej práce Mgr. Luďkovi Bártkovi, Ph.D. za odborné vedenie, metodickú pomoc a trpezlivosť pri vypracovávaní tejto práce.

Zhrnutie

Úlohou tejto práce bolo vyriešit' problém zadávania voľného textu pomocou metódy multi-tap a vypracovat' vstupný modul pre interpreter JVoiceXML. Súčasťou praktickej časti je modul a jednoduchá aplikácia, ktorá modul používa.

Kľúčové slová

JVoiceXML, VoiceXML, modul, multi-tap, Java

Obsah

Kapitola 1

Úvod

S príchodom nových technológií vznikajú stále vyššie a vyššie požiadavky na ich funkcionalitu a možnosti obsluhy. Zatiaľ čo funkcionalita sa vyžaduje najvyššia, obsluha by mala byť čo najjednoduchšia. Zaistenie intuitívneho ovládania pre užívateľa vyplýva, okrem pohodlia, aj zo znižovania nákladov firiem, ktoré by išli na zamestnanecké školenia. Sľubnú perspektívu do budúcnosti predstavuje ovládanie systémov hlasom, ktoré môže nahradiť klávesnicu vo väčšine prípadov. Dlho však efektívny prostriedok neexistoval.

V roku 2004 sa dostal do odporúčaní konzorcia *W3C* [38] značkovací jazyk *VoiceXML* [39]. Bol vytvorený na utváranie audio dialógov, ktoré zahrňujú syntetizovanú reč, digitalizovaný zvuk, rozlíšenie hovoreného a *DTMF* [5] vstupu, ako aj nahrávanie hovoreného vstupu, telefonovanie a zmiešané iniciatívne rozhovory.

VoiceXML je však iba značkovací jazyk. Preto bolo nutné vyvynúť programy, ktoré vedia *VoiceXML* dokument spracovať a vykonať príslušnú syntézu reči. Okrem iných komerčných aj voľných vznikol intepreter *JVoiceXML* [11]. Jeho hlavným cieľom je ponúknuť spracovanie *VoiceXML* všetkým užívateľom. Jednak preto, že je voľne šíriteľný a jednak preto, že je platformovo nezávislý.

S platformovou nezávislosťou súvisí, okrem použitia v počítačoch, aj použitie v telefónoch. Tu vzniká otázka vytvárania dokumentov, a teda písania textu. Práca navrhuje riešenie tejto otázky a vytvára vstupný modul pre *JVoiceXML*.

Prvá kapitola obsahuje bližší pohľad na *JVoiceXML*. Približuje definíciu *JVoiceXML*, jeho vlastnosti, rozoberá technológie a software nevyhnutný a doporučovaný pre prácu s ním. Ďalej popisuje základnú architektúru, podľa ktorej je *JVoiceXML* vytvorený a na ktorej vykonáva špecifické funkcie. Záver kapitoly vysvetľuje niektoré dôležité triedy.

Kapitola druhá približuje metódu *multi-tap* [14]. Venuje sa jej vlastnostiam a opisuje jej funkciu.

Tretia kapitola rozoberá požiadavky na implementáciu vstupného mo-

dulu. Venuje sa riešeniam častí modulu a problémom, ktoré nastali. Definuje klávesnicu vstupu a rozdelenie znakov na nej. Rozoberá spôsob zadávania vstupu, ktorý modul spracováva, opisuje proces zmeny číslného vstupu na text a zaoberá sa okrajovými udalosťami, ktoré by mohli ohroziť správnu funkcionalitu modulu.

Štvrtá kapitola obsahuje podrobný popis zdrojového kódu. Venuje sa ako modulu, tak aj demo aplikácii. Postupne rozoberá jednotlivé triedy a zvlásť popisuje ich metódy. Vysvetľuje dianie v každej metóde a ukazuje konkrétny spôsob implementácie a vyriešenia problému z návrhu. Napokon v krátkosti vysvetlí fungovanie demo aplikácie.

Príloha obsahuje zoznam súborov na priloženom CD nosiči.

Kapitola 2

JVoiceXML

Táto kapitola opisuje základné vlastnosti interpreteru *JVoiceXML*, jeho architektúru, nástroje nevyhnutné pre prácu s ním a podáva prehľad niektorých dôležitých tried.

2.1 Popis JVoiceXML

JVoiceXML je voľne dostupná implementácia jazyka *VoiceXML* verzie 2.1 [40] napísaná v programovacom jazyku *Java* [33]. Ponúka knižnicu pre jednoduchú tvorbu *VoiceXML* dokumentov a interpreter pre spracovanie *VoiceXML* dokumentov použitím štandardných API, ako *JTAPI* [36] a *JSAPI* [34] [20].

Projekt sa nachádza na hostingu *SourceForge* ako *open source*. Vyvýjaný je pod licenciou *GNU Library or Lesser General Public License (LGPL)* [8]. Cieľom autorov je aplikácia nezávislá na operačnom systéme, ktorá môže byť používaná voľne.

2.2 Používané technológie a software

Keďže projekt je vyvýjaný v Jave, pre užívateľa je dôležité mať nainštalované *JDK*. Väčšina kódu *JVoiceXML* používa súčasti verzie 1.5, a preto *J2SE 1.5* [35] je minimálna požiadavka.

Nemenej podstatný je *Ant*, ktorý je potrebný pre spustenie prehliadača a tvorbu binárnych kódov pre klientov. Odporúčaná verzia je aspoň 1.7.0 [6].

Ak si chce užívateľ prehliadnuť kódy, alebo angažovať sa do vývoja *JVoiceXML*, potrebný je editor, najlepšie však *Java IDE* [9]. *IDE* na rozdiel od obyčajného editora ponúka mnoho výhod, ktoré uľahčujú programátorovi prácu. Väčšina *IDE* má vstavaný *Ant*, avšak vhodnejšie je mať *Ant* oddelený kvôli dodržaniu nezávislosti na *IDE*.

Niektoré demo programy vyžadujú *servlet kontajner* [21]. Pre tento účel

môže poslúžit' voľne dostupný aplikačný server *Tomcat* [7] od *Apache Software Foundation* [1], alebo ktorýkoľvek iný.

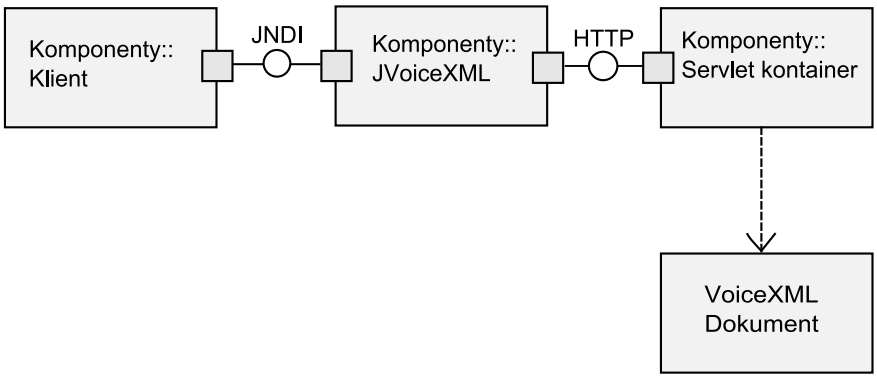
2.3 Štruktúra JVoiceXML

Základná štruktúra z pohľadu architektúry je na obrázku 2.1.

JVoiceXML sa spúšťa ako samostatný server a získava *VoiceXML* dokumenty zo *servlet kontajnera*. K nemu pristupuje pomocou protokolu, napríklad HTTP [37]. *VoiceXML* dokumenty su v *servlet kontajneri* uskladnené.

Klient pristupuje k *JVoiceXML* pomocou *Java Naming and Directory Interface (JNDI)* [32]. Klient môže takisto spustit' telefonický hovor pre aplikáciu pomocou tejto technológie [20].

Obr. 2.1: Architektúra JVoiceXML



2.4 Niektoré dôležité triedy

JVoiceXML obsahuje, popri množstve tried zabezpečujúcich širokú škálu funkcií, triedy, bez ktorých sa nezaobídeme ani pri programovaní tej najjednoduchšej aplikácie.

Niektoré z nich predstavíme na nasledujúcich riadkoch. Podrobný popis tried, rozhraní a ich metód je možné nájst' v *Java API* dokumentácii projektu *JVoiceXML* [4].

Balík `org.jvoicexml.xml.vxml` – Tento balík osahuje triedy dôležité pre vytváranie *VoiceXML* dokumentov. Keďže *VoiceXML* má presne stanovené elementy, ktoré môže obsahovať, každý takýto element je reprezentovaný triedou v jazyku *Java*. Triedy rozširujú triedu `AbstractXmlNode` z balíka `org.jvoice.xml`. Pri práci s elementami môžeme pri bližšom pohľade vidieť podobnosť s rozhraním *DOM* z balíka `org.w3c.dom` distribúcie *Java SE* [24].

Trieda `AbstractXmlNode` – implementuje rozhranie `XmlNode`, ktoré rozširuje rozhranie `org.w3c.dom.Node`, a teda táto trieda implementuje aj toto rozhranie. Z názvu vyplýva, že ide o abstraktnú triedu, konkrétne slúži ako základ pre všetky uzly v dokumente.
Obsahuje metódy pre prácu s uzlami a elementami. Keďže niektoré metódy sú implementáciou rozhrania `Node`, popíšeme iba niektoré rozširujúce.
Metóda `addChild` s parametrom `String tagName` pridá aktuálnemu uzlu potomka s názvom `tagName`.
Metóda `getAttribute` vracia hodnotu atribútu zadaného v parametre. Túto hodnotu vracia ako reťazec.
Metóda `getChildren` vracia kolekciu potomkov aktuálneho uzla.
Metóda `getNode` vracia zapúzdrený uzol.
Metóda `getTagName` vracia meno aktuálneho uzlu.

Trieda `VoiceXmlDocument` – Nachádza sa v podbalíku `vxml`. Predstavuje *VoiceXML* dokument podľa špecifikácií konzorcia *W3* [38]. Objekt tejto triedy môže dokument *VoiceXML* vytvoriť zavolaním konštruktora, alebo ho môže spracovať volaním metód. Pri vytvorení nového dokumentu sa vytvorí koreňový element < vxml >.
Metóda `getVxml` vráti koreňový element, čím umožníme pracovať s jeho potomkami.

Trieda `Vxml` – Je z rovnakého balíka a predstavuje koreňový element dokumentu *VoiceXML*. Jeho následníci môžu byť iba takzvané *dialógy*, čo predstavujú triedy `Form` a `Menu`.

Trieda `Form` – Reprezentuje *VoiceXML dialóg* < form >. Rozširuje triedu `AbstractXmlNode` o metódy slúžiace na vytvorenie potomkov < field > a < var >, nastavenie a získanie *Id* a získanie kolekcie jeho atribútov.

Trieda `RtpPlayer` – Predstavuje *RTP* prehrávač, ktorému sa pri vytvorení musí predať *RTP port* a *RTP control port* [19]. Prehrávač má okrem iného metódy pre jeho otvorenie a uzavretie.

Trieda `RtpRemoteClient` – Slúži na konfiguráciu *RTP streamingu* [22]. V konštruktore dostáva unikátne identifikátory pre kontrolu hovoru, systémový výstup, užívateľský vstup a čísla portov pre *RTP streaming* a kontrolnú informáciu.
Metódy slúžia na získanie *IP adresy* [10], *IP portu pre RTCP komunikáciu* a *IP portu pre RTP komunikáciu*.

Trieda `Session` – Rozhranie pre *session* (sedenie). Začína, keď užívateľ interaguje s kontextom *VoiceXML* interpretera, pokračuje pri nahrávaní a spracovávaní dokumentov a končí, ak je to vyžadované užívateľom, dokumentom, alebo kontextom intepreteru.

2.5 Moduly JVoiceXML

Jazyk *Java*, v ktorom je interpreter napísaný je objektovo orientovaný, čo umožňuje veľmi jednoduché pridanie modulu do aplikácie. V aplikácii stačí vytvoriť novú inštanciu triedy, ktorá tvorí modul. Vďaka tomu sa sprístupnia všetky metódy, ktoré modul obsahuje.

Kapitola 3

Multi-tap

Multi-tap [14] je metóda písania textu väčšinou využívaná mobilnými telefónmi.

S príchodom displejov na trh a ich zakomponovaním do telefónnych aparátov bolo potrebné zadávať text. V tej dobe už existovali klávesnice k počítačom, avšak ich použitie aj pri telefónoch by bolo veľmi nepraktické. Z toho dôvodu bolo cieľom dodržanie pôvodného konceptu mobilných telefónov – prenositeľnosť a jednoduché uskladnenie.

Métóda *multi-tap* využíva iba klávesnicu telefónu s počtom kláves dvanásť. Používané písmená sú z anglickej abecedy bez diakritiky. Na klávesnici sú rozdelené v skupinách po tri začínajúc klávesou '2' s výnimkou kláves '7' a '9', ktoré majú pridelené písmená štyri.

Klávesa '1' slúži na zadávanie matematických, interpunkčných alebo menových symbolov. Klávesy '0' a '1' môžu mať pridelené znaky iné, prípadne sú medzi ne rozdelené. Klávesa '$*$' slúži väčšinou na prepínanie na veľké písmená alebo/a na číslice. Funkcia kávesy '#', ako aj ostatných, záleží na výrobcovi telefónu.

Samotné zadávanie textu sa vykonáva pomocou stláčania kláves. Jedno stlačenie klávesy znamená zadanie prvého symbolu priradeného k danej klávese. Viacnásobné stlačenie rovnakej klávesy zabezpečí zadanie znaku na ďalších pozíciách klávesy vzhľadom na počet jej stlačení.

Príslušný znak sa bez možnosti zmeny[1] zapíše, ak je následne stlačená iná klávesa, alebo po prečkaní časového úseku.

Zadanie viacerých písmen z rovnakej klávesy riešia telefóny vypršaním určitého času po stlačení klávesy, alebo nutnosťou stlačiť inú klávesu – typicky šípku doprava.

Niektoré telefóny podporujú aj diakritiku pridaním diakritických písmen navyše. Napríklad pre klávesu '2' sa za *a, b, c, 2* pridajú *á, ä, č*.

Multi-tap sa najviac využíva na písanie textových správ SMS, vyplňova-

1. okrem zmazania znaku

nie informácii o kontakte v telefóne, chatovanie na internete, alebo zadanie mena hráča v mobilnej hre.

V súčasnej dobe je vidieť postupný ústup telefónov s *multi-tap*. Je to spôsobené najmä zdokonaľovaním mobilných technológií, s čím súvisí aj čoraz častejší výskyt telefónov s *QWERTY* klávesnicou [17] pri zachovaní ich pôvodnej veľkosti. Na trh sa dostávajú aj telefóny s dotykovým displejom. A nie sú to len telefóny, ale aj *PDA* [15], ktoré sú v rozmanitosti funkcií lepšie ako telefóny.

Zatiaľ však tieto prístroje nie sú cenovo dostupné všetkým, a preto potrvá nejaký čas, kým telefóny s metódou *multi-tap* úplne vymiznú.

Kapitola 4

Rozbor požiadaviek na implementáciu

Táto kapitola sa zaoberá požiadavkami na správnu implementáciu modulu. Rozoberá spôsob vstupu, samotné fungovanie modulu a objasňuje uprednostnenie tohto riešenia pred riešeniami vyplývajúcimi z iného návrhu. Kapitola uvádza aj niekoľko problémov, ktoré súvisia s výberom aktuálnej implementácie a s ktorými sa modul musí vysporiadať.

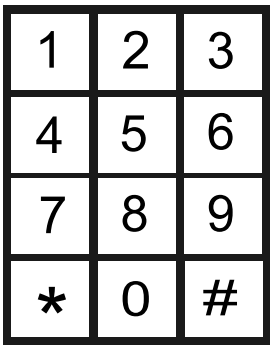
4.1 Klávesnica vstupu

Implementácia predpokladá klasické rozloženie numerickej klávesnice mobilného alebo pevného telefónu s dvanástimi tlačidlami – štyri riadky po tri tlačidlá.

Tlačidlo '1' je umiestnené v ľavom hornom rohu. Nasledujúce klávesy pokračujú smerom doprava, po zaplnení riadku je ďalšia klávesa v nasledujúcom riadku vľavo. Tlačidlo '$*$' je v ľavom dolnom rohu, '#' v pravom dolnom. Tlačidlo '0' sa nachádza v dolnom riadku medzi nimi (obr. 4.1).

Pre nutnosť písania voľného textu s čo najväčšími možnosťami sú

Obr. 4.1: Klávesnica vstupu



využité všetky tlačidlá okrem tlačidla '∗'. Podrobné rozdelenie znakov popisuje kapitola 4.2.

4.2 Mapa znakov

Mapovanie písmen na čísla je podobné, ako u mobilných telefónov. Použité sú písmená bez diakritiky. Avšak ich doplnenie je len otázkou pridania znakov do mapy.

Ako predloha k mapovaniu písmen abecedy bol použitý telefón značky *NOKIA*. Dôvod jeho použitia je ten, že značka *NOKIA* je vo svete rozšírená a najviac známa. V podstate je možné použit' aj iné telefóny na európskom trhu, ako napríklad *SIEMENS, ERICSSON, LG, SAMSUNG*.

Pre ďalšie znaky, ako medzera, plus, atď . . . , bolo nutné použit' vlastnú implementáciu. Neexistuje štandard, ktorý by definoval rozloženie týchto znakov a každý telefón používa rozloženie iné. Tabuľka 4.1 udáva prehľad rozloženia znakov na tlačidlách klávesnice.

Ak je v riadku uvedený znak '–', jedná sa o znak pomlčka a bude vypísaný. Ak je tento symbol bez úvodzoviek, znamená to, že na danej pozícii nie je definovaný žiadny symbol pre určenú klávesu.

Tabuľka 4.1: Tabuľka znakov pridelených klávesam podľa počtu stlačení klávesy

	Znak na pozícii						
Klávesa	1.	2.	3.	4.	5.	6.	7.
1	' '	'.'	','	'?'	'!'	'–'	'1'
2	'a'	'b'	'c'	'2'	-	-	-
3	'd'	'e'	'f'	'3'	-	-	-
4	'g'	'h'	'i'	'4'	-	-	-
5	'j'	'k'	'l'	'5'	-	-	-
6	'm'	'n'	'o'	'6'	-	-	-
7	'p'	'q'	'r'	's'	'7'	-	-
8	't'	'u'	'v'	'8'	-	-	-
9	'w'	'x'	'y'	'z'	'9'	-	-
0	'+'	'='	'>'	'<'	'0'	-	-

4.3 Zadávanie vstupu

Pre účely modulu sme zvolili zadávanie symbolov reprezentovaných ako signály *DTMF* [5] na štandardný vstup.

Prvý návrh spočíval v zadávaní vstupu po jednej číslici. Po zadaní čísla by modul čakal na ďalšie stlačenie čísla určitý čas a po jeho uplynutí by sa skonvertovalo na príslušný symbol.

Takéto riešenie sa stretlo s neúspechom, a to vďaka povahe čítania zo štandardného vstupu v jazyku *Java*, ktorý sa priebežne ukladá do vyrovnávacej pamäte operačného systému. Po zavolaní metódy `System.in.read()` [31] sa prvý symbol prečíta a čaká sa na ďalšie zavolanie metódy `read()` [3]. Takto by užívateľ za každým stlačením klávesy musel vstup odoslať, čo nie je praktické.

Preto modul pracuje takisto s bufferovaným vstupom a výstupom, kde sa naraz načíta celý vstup. Pre užívateľa to znamená zadávanie celej postupnosti číslic, ktorá sa po stlačení klávesy **Enter** predáva modulu. Užívateľ, navyše, pri stláčaní rovnakej kávesy nie je limitovaný počtom znakov pridelených tejto klávese. Pri jeho prekročení sa sekvencia znakov pre klávesu opakuje.

4.4 Konverzia číslic na znaky

Po zadaní vstupu sa zadaný reťazec číslic rozdelí na jednotlivé symboly, ktoré sa uložia lokálne. Tieto symboly sa postupne pridávajú do vstupného buffera modulu. Ak sa pridávaný symbol zhoduje s posledným pridaným, tak ho pridá a zvýši počítadlo rovnakých číslic o jedna. Ak sa nezhoduje, prebehne konverzia aktuálnych symbolov vo vstupnom bufferi.

Samotná premena týchto symbolov sa deje podľa mapy kláves, pričom počet stlačení klávesy je rovný aktuálnemu stavu počítadla stlačení, ktoré sa aktualizuje pri pridaní symbolu do buffera. Z predchádzajúcich vlastností je zrejmé, že obsah buffera sa skonvertuje na jedno písmeno alebo iný symbol výsledného textu, ktorý sa uloží do buffera výstupného.

Po konverzii sa obsah vstupného buffera vyčistí a symbol, ktorý „aktivoval“ konverziu sa do neho môže bezpečne uložiť.

Modul prejde k ďalšiemu prvku poľa symbolov a proces sa opakuje. Vďaka tomuto chovaniu pridávania nových symbolov a vyčisteniu buffera po konverzii je zaručené, že vstupný buffer aktuálne obsahuje iba symboly rovnaké, teda jedna klávesa bola stlačená viac krát po sebe, alebo iba raz. Taktiež máme zabezpečené, že po tejto postupnosti bola stlačená klávesa iná, a teda konverzia prebieha korektne.

4.5 Zadanie dvoch písmen po sebe z rovnakého tlačidla

Doterajší popis zadávania voľného textu pomocou číslic je postačujúci iba pre istú množinu slov. Problém nastáva, ak užívateľ chce napísať slovo, v ktorom sa nachádzajú dve po sebe idúce písmená, ktoré sú namapované na rovnakú klávesu (napr. *Hello*).

Ako je uvedené v časti 4.3, povaha štandardného vstupu jazyka *Java* obmedzila možnosť postupného zadávania a zobrazovania vstupu a spolu s tým sa riešenie problému pomocou časových úsekov stalo nezrealizovateľným. Spočívalo by v ustanovení určitého času, do ktorého musí užívateľ stlačiť rovnakú klávesu, ak chce získať nasledujúci znak prislúchajúci danej klávese. Ak by to do tohto času nestihol, už by predchádzajúci znak textu nemohol zmeniť inak, ako jeho zmazaním.

Súčasná implementácia vychádza z nášho predpokladu pre bufferovaný vstup pre modul. Jej riešenie pozostáva zavedením symbolu, ktorý nemá v mape kláves pridelené svoje znaky, a teda nemôže byť premenený na text. Na tento účel je vyčlenená klávesa '#'. Zadaním symbolu '#' užívateľ dáva najavo, že chce, aby ďalšie stlačenie rovnakej klávesy, ako pred symbolom '#', bolo zaznamenané ako nové písmeno textu. V „reči" modulu '#' nerobí nič iné, ako explicitne volá konverziu symbolov vo vstupnom bufferi.

Kapitola 5

Implementačná časť

Táto kapitola sa venuje podrobnému popisu zdrojového kódu modulu. Podáva detailný rozbor diania v module na úrovni metód a poskytuje prehľad postupnosti operácií, ktoré modul vykonáva, aby užívateľ dostal na výstup korektný text.

5.1 Spôsob implementácie

Modul je implementovaný v jazyku *Java*. Podmienka na programovací jazyk bola daná tým, že samotný interpreter *JVoiceXML* je napísaný v *Jave*.

Tento jazyk poskytuje výhody, ktoré značne zjednodušujú implementáciu. Ako príklad uvedieme použitie triedy `StringBuilder` [30] namiesto `String` (kapitola 5.2.2) alebo `ArrayList` [25] namiesto obyčajného poľa (kapitola 5.2.1). V neposlednom rade nám *Java* umožňuje elegantné použitie modulu v demo aplikácii.

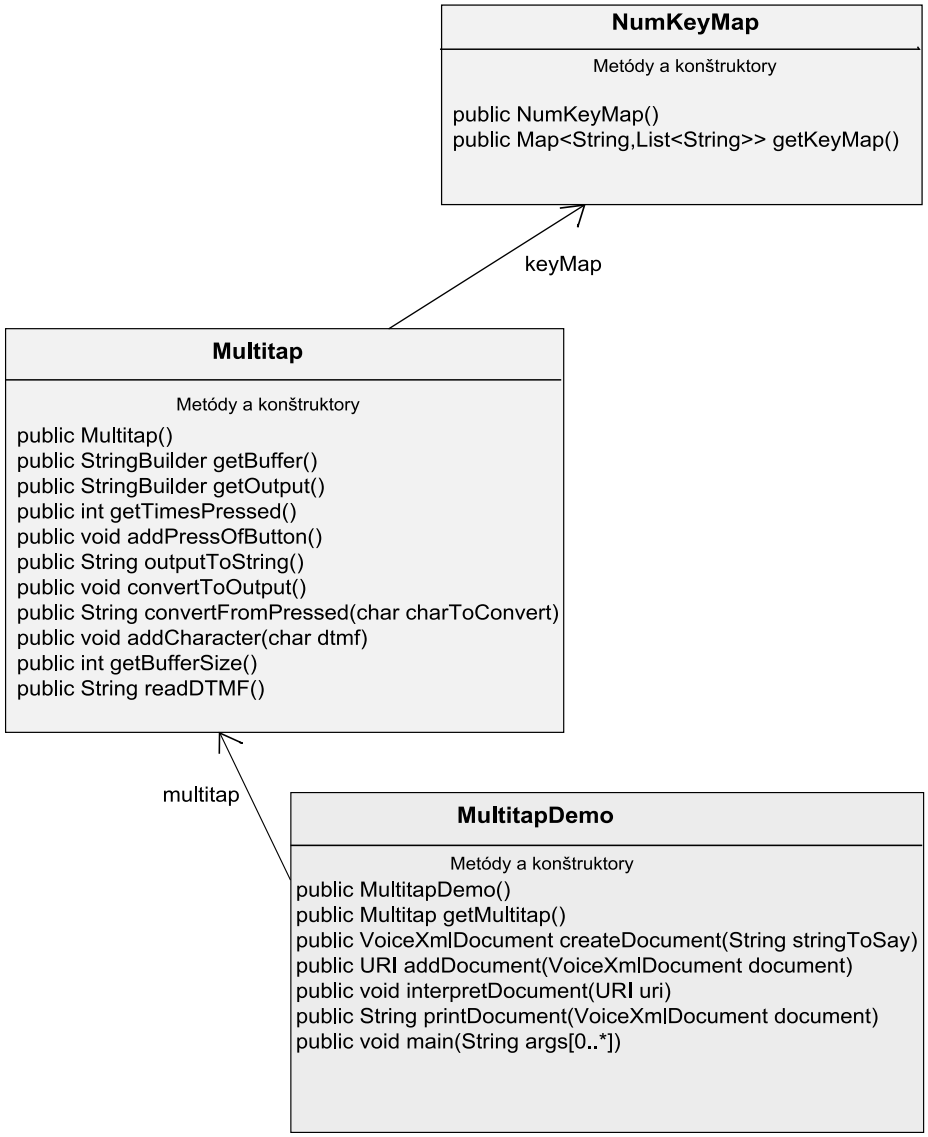
5.2 Popis tried modulu

Samotný modul môže byť napísaný v jednom súbore, ktorý by obsahoval dve triedy, alebo jednu samostatnú triedu s atribútom typu `Map` [27], ktorý by hneď obsahoval sekvenciu znakov. Obe z týchto riešení sú správne. Avšak dôležitým prvkom je aj prehľadnosť kódu, čo by predchádzajúce riešenia značne obmedzovali. Preto modul obsahuje dve samostatné triedy – `NumKeyMap` a `Multitap`. Trieda `MultitapDemo` nepatrí do modulu, ale predstavuje nezávislú demo aplikáciu, v ktorej je modul na ukážku použitý. Obrázok 5.1 zobrazuje diagram tried modulu a demo aplikácie.

5.2.1 Trieda `NumKeyMap`

Trieda `NumKeyMap` je verejná trieda. Nevykonáva žiadnu funkcionalitu modulu, ale je jeho neoodeliteľnou súčasťou. Obsahuje mapu znakov dôležitých pre konverziu z číselného vstupu.

Obr. 5.1: UML diagram modulu a demo aplikácie



V implementácii sa ponúka možnost použiť typ `Enum`, ktorý by atribútu – klávese – priradil množinu konštánt – znakov. Pri tomto spôsobe nastávajú problémy pri výbere správneho prvku. Bolo by nutné napísať niekoľko pomocných metód, ktoré by výber správneho znaku uľahčovali. Preto je vhodnejšie použiť mapu, ktorá má jednoduchý spôsob dotazovania na hodnoty uložené pod kľúčom.

„Konverznú tabuľku“ v triede predstavuje inštancia hešovanej mapy `HashMap`, ktorá sa uchováva v súkromnom atribúte triedy `keyMap` typu `Map`. Do mapy sa pod klúčom typu `String` ukladajú hodnoty typu `List<String>`. To znamená, že k symbolu z klávesnice reprezentovanému ako reťazec sa priradí zoznam reťazcov, a teda symbolov, ktoré pôjdu do výsledného textu. Zatiaľ máme iba prázdnu mapu, o ktorej vieme, čo a ako do nej ukladať. Vlastné uloženie konkrétnych prvkov vykonáva konštruktor `NumKeyMap`.

Konštruktor `NumKeyMap()`

Samotný konštruktor vykonáva aj vkladanie prvkov do mapy. To znamená, že po vytvorení inštancie tejto triedy sa vytvorí aj kompletná mapa kláves.

Na začiatku sa vytvorí prázdna inštancia zoznamu `listOfValues` typu `ArrayList<String>`. Pre naše potreby je `ArrayList` najvhodnejší, pretože používa priamy prístup k jeho prvkom.

Po vytvorení prázdneho zoznamu sa do neho pridajú reťazce veľkosti jedna (znaky na výstup) v poradí, v akom ich chceme zobrazovať po stlačení klávesy.

Po pridaní všetkych hodnôt sa celý zoznam uloží do mapy `keyMap` pod kľúčom reprezentujúcim klávesu. Kvôli vyhnutiu sa problémom s možnou zámenou klávesy a iného reťazca sú použité kľúče ako jednoznačný viacznakový reťazec. Napríklad klávesu `'1'` reprezentuje kľúč `"one"`, klávesu `'2'` kľúč `"two"`, atď . . .

Po vložení kľúča a jeho hodnoty do mapy sa obsah zoznamu `listOfValues` zmaže a pokračuje sa s pridávaním hodnôt pre nasledujúcu klávesu. Takýmto spôsobom vloží všetky hodnoty a vytvorí žiadanú mapu kláves.

Tento spôsob ma výhodu v tom, že modulu stačí, aby si vytvoril jednu inštanciu triedy `NumKeyMap` a spolu s ňou sa vytvorí iba jedna mapa kláves, čím neprichádza k zbytočne opakovanej tvorbe rovnakej mapy.

Metóda `getKeyMap`

Trieda `NumKeyMap` obsahuje jedinú metódu - `getKeyMap()`. Je to verejná metóda, pomocou ktorej môže modul získať mapu znakov.

5.2.2 Trieda `Multitap`

Táto trieda predstavuje základnú funkcionalitu modulu. Obsahuje všetky potrebné metódy na prečítanie vstupu, zmenu na text a vrátenie výsledku. Trieda je verejná a implementuje rozhranie `RemoteCharacterInput` [18] z balíka `org.jvoicexml.client.jndi`, ktoré predpisuje metódu `addCharacter`.

Modul pracuje s niekoľkými základnými štruktúrami, v ktorých si uchováva spracovávané dáta. Tieto štruktúry sú v triede reprezentované atribútami.

Atribút `timesPressed` vyjadruje počet stlačení klávesy a reprezentuje ho celé číslo väčšie ako nula.

Atribút `buffer` značí vstupný buffer, z ktorého sa vstupné symboly konvertujú na príslušný znak.

Atribút `output` značí výstupný buffer, v ktorom sa hromadí text, ktorý pôjde na výstup ako výsledok. Oba atribúty `buffer`, `output` sú typu `StringBuilder`. Existuje aj možnosť použiť `StringBuffer` [29], avšak táto trieda je vláknovo bezpečná, a preto by mala byť použitá pri multivláknových aplikáciách.

`StringBuilder` je oproti obyčajnému `String` vhodnejší, pretože modul postupne tvorí text z častí. V triede `String` má operácia zlučovania reťazcov veľkú réžiu a pri viacnásobnom zlučovaní to môže spôsobiť spomalenie aplikácie. Naopak, trieda `StringBuilder` je navrhnutá tak, aby opakované zlučovanie reťazcov prebiehalo efektívne a rýchlo.

Proces konverzie na text vykonávajú metódy.

Metóda `addPressOfButton`

Metóda `addPressOfButton` je bez parametra. Má jedinú funkciu, a to zvýšenie stlačenia klávesy o jedna. Teoreticky táto metóda nie je nutná a zvýšenie o jednotku by sa mohlo vykonávať podľa potreby priamo pomocou `timesPressed++`. Avšak realizácia tejto operácie pomocou samostatnej metódy znižuje pravdepodobnosť chyby v kóde.

Metóda `addCharacter`

Táto metóda je predpísaná rozhraním `RemoteCharacterInput` a má parameter `dtmf` typu `char`. Rozhranie popisuje túto metódu, ako metódu pre detekciu vstupu symbolov a práve to je pre tento modul dôležité.

Vo vnútri metódy sa deje niekoľko jednoduchých vecí. Na začiatku skontroluje, či sa hodnota parametra rovná znaku '#'. Ak nie, pridá hodnotu parametra do atribútu `buffer` a zvýši čítač `timesPressed` o jedna. Znamená to, že znak '#' sa nenachádza medzi symbolmi pripravenými na konverziu.

Metóda `convertToOutput`

Táto bezparametrická metóda vykonáva úkony tesne pred konverziou symbolov, dáva „pokyn" na konverziu a stará sa o aktualizáciu tesne po vykonaní konverzie.

Najprv zistí, aký je posledný symbol vo vstupnom bufferi. Následne spustí pomocnú metódu `convertFromPressed`, ktorej predá zistený symbol a jej výsledok pridá do výsledku `output`. Keďže v atribúte `buffer` sa nachádzajú iba rovnaké symboly, metóde `convertFromPressed` stačí predať iba jeden. Po konverzii sa zmaže obsah `buffer` a vynuluje čítač `timesPressed`.

Metóda `convertFromPressed`

Táto metóda vykonáva samotnú konverziu. Má jeden parameter typu `char`, v ktorom metóda dostáva symbol, podľa ktorého má vykonať zmenu. Vracia `String`, no v skutočnosti sa jedná iba o reťazec veľkosti jedna.

Na začiatku metódy je vytvorená lokálna premenná `outputOneChar` typu `String`, v ktorej bude uložený výsledok metódy. Nasledujú pomocné premenné – zoznam `listOfOutput` a celé číslo `position` inicializované na nulu.

Väčšina metódy je tvorená sériou podmienok vzájomne výlučných. Každá podmienka overí, ktorý symbol z množiny {0,1,2,3,4,5,6,7,8,9} symbolov vstupnej klávesnice sa zhoduje so symbolom v parametre. Pri zhode sa do `listOfOutput` uloží hodnota uložená v mape `keyMap`, teda zoznam znakov pre danú klávesu.

Do premenej `position` sa uloží pozícia, na ktorej sa nachádza správny znak v zozname `listOfOutput`. Vypočíta sa tak, že na počet stlačení klávesy mínus jedna sa aplikuje operácia modulo počet znakov v `listOfOutput`. Samotný počet stlačení `timesPressed` nestačí, pretože

užívateľ môže zadať viac stlačení rovnakej kávesy, ako je počet znakov jej pridelených. Tento prípad rieši práve operácia modulo, vďaka ktorej nie je možné „vybehnúť" mimo rozsah zoznamu.

Zníženie hodnoty v atribúte `timesPressed` o jedna je vynútené indexovaním v štruktúre `List`, ktoré sa začína od nuly. Bez dekrementácie by operácia modulo vracala nesprávne hodnoty, a to cyklicky posunuté o jedno miesto dopredu (napríklad pri dvoch stlačeniach '2' by sme dostali 'c', pri štyroch stlačeniach 'a').

V tele podmienky sa ako posledné vykoná uloženie hodnoty zo zoznamu `listOfOutput` na pozícii `position` do lokálnej premennej `outputOneChar`, ktorú metóda vráti ako svoj výsledok.

Metóda `readDTMF`

Táto metóda vykonáva réžiu potrebnú pre správny beh modulu od načítania zo štandardného vstupu po správne vykonanie konverzie. Vracia výsledok typu `String`, a teda hotový text, ktorý môže byť predaný napríklad na štandardný výstup (kapitola 5.3).

Na začiatku metódy vytvoríme `Reader` [28] štandardného vstupu a ten použijeme na vytvorenie `BufferedReader` [26].

Po potvrdení sa do premennej `dtmfString` uloží celý vstup ako reťazec. Potrebujeme z neho získať jednotlivé znaky. To urobíme pomocou metódy `String.toCharArray()`, ktorá prevedie reťazec na pole symbolov.

Nasleduje cyklus, v ktorom metóda prechádza všetky symboly v poli a ďalej spracováva. Ak vybraný symbol je '#', alebo ak vstupný buffer nie je prázdny a tento symbol sa nezhoduje s posledným v bufferi, zavolá sa metóda `convertToOutput`. Tým sa vykoná potrebná konverzia, pridá sa časť k výsledku a zmaže buffer. Následne je tento symbol pridaný do prázdneho buffera. Ak vybraný symbol podmienku nesplňuje, je do buffera pridaný hneď.

Keďže možná konverzia sa deje ešte pred pridávaním, problém nastáva na konci vstupu, kde posledné rovnaké symboly nie sú zmenené. Toto rieši posledná podmienka v tele metódy, a to test, či cyklus dosiahol posledný symbol. Pozitívna odpoveď znamená konverziu toho, čo sa nachádza v bufferi. Pri negatívnej odpovedi sa nedeje nič.

Ak by sa v metóde nenachádzala táto podmienka, musel by užívateľ ukončiť svoju vstupnú sekvenciu symbolom '#' a to už nie je ani intuitívne, ani pohodlné.

Môžme si všimnúť, že v tele metódy sa nachádza `try-catch` blok,

ktorý zachytáva výnimku `IOException`. Deje sa tak z toho dôvodu, že metóda `BufferReader.readLine()` túto výnimku vyhadzuje. Po jej zachytení sa zapíšu informácie do logu a aplikácia sa ukončí.

Pomocné metódy

Trieda `Multitap` obsahuje aj niekoľko pomocných metód, ktoré nie sú nevyhnutné pre správnu funkciu modulu, ale zvyšujú prehľadnosť kódu, znižujú pravdepodobnosť programátorskej chyby a pomáhajú triede dodržať konvencie programovania v jazyku *Java*. Sú to metódy:

`getBuffer` – vracia vstupný buffer typu `StringBuilder` – `buffer`

`getOutput` – vracia výstupný buffer typu `StringBuilder` – `output`

`getTimesPressed` – vracia počet stlačení rovnakej klávesy `timesPressed`

`outputToString` – vracia reťazec, ktorý obsahuje `output`

`getBufferSize` – vracia aktuálnu dĺžku reťazca v `buffer`

5.3 Popis demo aplikácie

Na ukážku použitia modulu v interpreteri JVoiceXML bol vytvorený demo program. Jedná sa o jednoduchú, v príkazovom riadku spustiteľnú aplikáciu, ktorá text napísaný metódou *multi-tap* interpretuje syntetizovaným hlasom.

Pre správne fungovanie našej demo aplikácie je nutné skopírovať adresár *MultitapDemo* do *$JVOICEXML_ HOME$ /demo*, kde *$JVOICEXML_ HOME$* je domovský adresár nainštalovaného interpreteru.

Pre spustenie, odporúčame dodržať postup opisujúci spustenie demo programov nachádzajúcich sa v inštalácii *JVoiceXML* [20].

5.3.1 Trieda `MultitapDemo`

Táto trieda je jediná, ktorá tvorí samostatnú demo aplikáciu využívajúcu vytvorený modul. Väčšina použitého kódu je z dema *HelloWorldDemo*, ktoré je taktiež súčasťou inštalácie *JVoiceXML* a nachádza sa v *$JVOICEXML_ HOME$/demo*.

Metódy `addDocument`, `interpretDocument` a `printDocument` zostávajú rovnaké, ako v *HelloWorldDemo*. Jemná zmena je v konštruktore, v metóde `createDocument` a v pridaní novej metódy `getMultitap`.

V konštruktore triedy vytvoríme novú inštanciu triedy `Multitap`, čím sprístupníme metódy modulu. Taktiež sa vytvorí nový kontext [23], ktorým sa sprístupnia *JNDI* zdroje, ktoré sú nutné pre komunikáciu s hlasovým prehliadačom *JVoiceXML*.

Metóda `createDocument` je pozmenená tým, že jej bol pridaný parameter. V tejto metóde sa vytvára *VoiceXML* dokument, ktorý bude interpretovaný. Preto práve tu je potrebné pridať text získaný metódou *multi-tap* a práve ten sa predáva parametrom `stringToSay`.

Metóda `addDocument` pridáva dokument do lokálneho ukladacieho priestoru (*repository*). Jej parameter je ukladaný *VoiceXML* dokument, ako výsledok vracia *URI* [2], pod ktorou je dokument uložený. Metóda najprv v kontexte vyhľadá danú *repository* a uloží ju do premennej typu `MappedDocumentRepository`. Získa z nej *URI* a pridá dokument pod daným *URI* a názvom dokumentu.

Metóda `interpretDocument` spúšťa hlasový prehliadač, ktorý interpretuje dokument. Ako parameter dostáva *URI* interpretovaného dokumentu v *repository*. Z kontextu získa inštanciu `JVoiceXML` [12], ktorej rozhranie slúži ako vstupný bod pre všetkých vzdialených klientov. Metóda, ďalej, vytvorí nový *RTP* prehrávač s *RTP portom* číslo 4242 a *RTP kontrolným portom* číslo 4243. Nasleduje vytvorenie *RTP* vzdialeného klienta.

Port 4242 je najrozšírenejšie používaný na internete pre *MUD* [13] – druh komunikačného systému. Často je používaný ako príklad v ukážkach kódov, alebo ako alternatívny *HTTP port* [16]. Preto vybrané číslo nemá žiadny vplyv na našu demo aplikáciu.

Samotné spustenie hlasového prehliadača sa deje až vytvorením sedenia – *session*. Získava ho volaním metódy `createSession` na inštancii *JVoiceXML*. Pomocou `session.call(uri)` prehliadač vie, ktorý dokument má interpretovať. Po ukončení sedenia, sa sedenie aj prehrávač zavrú.

V metóde `main` najprv vytvoríme novú inštanciu `MultitapDemo`. Nasleduje získanie inštancie triedy `Multitap` a zavolanie metódy `readDTMF`. Táto metóda získa zo vstupu správny text (kapitola 5.2.2), ktorý sa dočasne uloží v premennej typu `String`. Reťazec sa potom predá ako parameter metóde `createDocument`, ktorá vytvorí *VoiceXML* dokument. Volaním metódy `printDocument` sa vypíše vytvorený dokument na štandardný výstup. Na záver sa volajú metódy `addDocument` a `interpretDocument`.

Kapitola 6

Záver

Prvou úlohou práce bolo navrhnúť riešenie problému pre zadávanie textu pomocou metódy *multi-tap*. Počas návrhu sa vyskytlo niekoľko problémov, kvôli ktorým bolo nutné návrh pozmeniť, alebo vybrať sa inou cestou. Všetky problémy sú v texte spomenuté zahŕňajúc stručný opis pôvodného riešenia.

Druhou úlohou bolo implementovať vstupný modul pre interpreter *JVoiceXML*, ktorý vykonáva metódu *multi-tap*. Výsledok je modul, ktorý korektne rieši zadaný problém, je v *JVoiceXML* použiteľný a je platformovo nezávislý.

Okrem modulu bola vytvorená aj jednoduchá aplikácia, ktorá využíva modul a potvrdzuje jeho funkčnosť a správnosť.

Súčasná implementácia modulu nie je jediná zo správnych. Ďalším možným riešením by bolo s prihliadnutím na širšiu využiteľnosť. Metódy by boli navrhnuté tak, aby umožňovali ich implementáciu ako v aplikácii so štandardným vstupom, tak aj v aplikácii s grafickým, prípadne webovým rozhraním. Tým by sa modul stal flexibilnejší a uspokojujúci širšie publikum užívateľov.

Kapitola 7

Obsah CD

Priložené CD obsahuje:

- balík s triedami
  - `NumKeyMap.java`
  - `Multitap.java`
  - `MultitapDemo.java`
- textovú časť práce vo formáte pdf
- zdrojový súbor textovej časti vo formáte LaTeX

Literatúra

[1] Apache software foundation.
http://www.apache.org.

[2] T. Berners-Lee. Uniform resource identifiers (uri): Generic syntax, August 1998.
http://www.ietf.org/rfc/rfc2396.txt.

[3] The ins and outs of standard input/output.
http://www.javaworld.com/javaworld/jw-03-2001/jw-0302-java101.html.

[4] Jvoicexml documentation.
http://jvoicexml.sourceforge.net/documentation.htm.

[5] Dual tone multiple frequency.
http://dtmf.org.

[6] Apache Software Foundation. Apache ant.
http://ant.apache.org/.

[7] Apache Software Foundation. Apache tomcat.
http://tomcat.apache.org.

[8] Gnu library or lesser general public license (lgpl).
http://jvoicexml.sourceforge.net/licence.htm.

[9] Java integrated development environment.
http://www.apl.jhu.edu/ hall/java/IDEs.html.

[10] Internet protocol address.
http://www.linfo.org/ip_address.html.

[11] Jvoicexml 0.6.
http://jvoicexml.sourceforge.net/.

[12] Interface jvoicexml.
http://jvoicexml.sourceforge.net/api-0.6/org.jvoicexml/org/
/jvoicexml/JVoiceXml.html.

[13] Multi-user dungeon.
http://www.livinginternet.com/d/d.htm.

[14] Multi-tap method.
http://en.wikipedia.org/wiki/Multi-tap.

[15] Personal digital assistent.
http://encyclopedia2.thefreedictionary.com/PDA.

[16] Port 4242.
http://www.iss.net/security_center/advice/Exploits/Ports/4242/
/default.htm.

[17] Qwerty keyboard.
http://home.earthlink.net/ dcrehr/whyqwert.html.

[18] Jvoicexml java class remotecharacterinput.
http://jvoicexml.sourceforge.net/api-0.6/org.jvoicexml/org/
/jvoicexml/client/jndi/RemoteCharacterInput.html.

[19] Real-time application protocol.
http://www.ietf.org/rfc/rfc3550.txt.

[20] Dirk Schnelle. Jvoicexml 0.6 user guide, June 2008.
http://prdownloads.sourceforge.net/jvoicexml/jvxml-userguide-
0.6.pdf?download.

[21] Servlet container.
http://www2.roguewave.com/support/docs/leif/leif/html/
/bobcatug/2-2.html.

[22] Real-time streaming protocol.
http://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol.

[23] SUN.
http://java.sun.com/javase/6/docs/api/javax/naming/
/Context.html.

[24] SUN. Document object model (dom).
http://java.sun.com/j2se/1.4.2/docs/api/org/w3c/dom/package-
summary.html.

[25] SUN. Java class arraylist.
http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html.

[26] SUN. Java class bufferedreader.
http://java.sun.com/javase/6/docs/api/java/io/
/BufferedReader.html.

[27] SUN. Java class map.
http://java.sun.com/javase/6/docs/api/java/util/Map.html.

[28] SUN. Java class reader.
http://java.sun.com/javase/6/docs/api/java/io/Reader.html.

[29] SUN. Java class stringbuffer.
http://java.sun.com/javase/6/docs/api/java/lang/
/StringBuffer.html.

[30] SUN. Java class stringbuilder.
http://java.sun.com/javase/6/docs/api/java/lang/
/StringBuilder.html.

[31] SUN. Java class system.
http://java.sun.com/javase/6/docs/api/java/lang/System.html.

[32] SUN. Java naming and directory interface.
http://java.sun.com/products/jndi.

[33] SUN. Java programming language.
http://java.sun.com.

[34] SUN. Java speech api 1.0 (jsapi).
http://java.sun.com/products/java-media/speech/forDevelopers/
/jsapi-doc/index.html.

[35] SUN. Java standard edition 1.5.
http://java.sun.com/javase/downloads/index.jsp.

[36] SUN. Java telephony api 1.2 (jtapi).
http://java.sun.com/products/jtapi/jtapi-
1.2/JTAPIWhitePaper_0_7.html.

[37] W3C. Hypertext tranfer protocol - http 1.1.
http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[38] World wide web consortium.
http://www.w3.org.

[39] W3C. Voice extensible markup language (voicexml) 2.0, 2004.
http://www.w3.org/TR/voicexml20/.

[40] W3C. Voice extensible markup language (voicexml) 2.1, 2007.
http://www.w3.org/TR/voicexml21/.