VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hlasový portál na platformě VoiceXML

Voice Portal with VoiceXML Platform

2014

Bc. Ondřej Ždych

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Ondřej Ždych**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T059 Mobilní technologie

Téma: Hlasový portál na platformě VoiceXML
Voice Portal with VoiceXML Platform

Zásady pro vypracování:

Nedílnou součástí komunikace v rámci webového portálu může být samoobslužný komunikační kanál. Cílem diplomové práce je implementace webového portálu s podporou VoiceXML.
1. Seznámení s problematikou VoiceXML.
2. Specifikace požadavků pro podporu automatického převodu textu do hlasu a obráceně(TTS a ASR).
3. Návrh webového portálu s možnostmi převedení textu do hlasu a automatického rozpoznání hlasu.
4. Zátěžové testy portálu.

Seznam doporučené odborné literatury:

James A. Larson *VoiceXML Introduction to Developing Speech Applications* Prentice Hall 2002, ISBN-13: 978-0130092625
Adam Hocek, David Cuddihy, *Definitive VoiceXML* Prentice Hall 2002, ISBN-13: 978-0130463456
Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Touto cestou bych rád poděkoval vedoucímu diplomové práce, panu Ing.Pavlovi Nevludovi, především za konzultace a odborné rady, které jsem zúročil při psaní tohoto textu.

Abstrakt

Diplomová práce se zabývá návrhem a implementací webového a hlasového portálu na platformě VoiceXML.

Nejdříve je představen samotný jazyk VoiceXML a vybrané hlasové platformy. Následuje úvod do problematiky související s technologiemi pro syntézu řeči (TTS) a automatické rozpoznávání řeči (ASR).

V práci je dále popsán návrh a implementace webového a hlasového rozhraní informačního systému pro rezervaci letenek. Nechybí zde ani popis konfigurace hlasového portálu.

Součástí práce bylo také provedení zátěžových testů navrženého hlasového portálu.

Klíčová slova: VoiceXML, VXML, hlasový portál, JSON, automatické rozpoznávání řeči, syntéza řeči, JavaScript, Node.js, AngularJS, Asterisk, VoiceGlue, Voxeo Prophecy, SIP, ASR, TTS

Abstract

Master's thesis deals with design and implementation of web and voice portal with VoiceXML platform.

There is an introduction to related technologies like speech synthesis (TTS) and automatic speech recognition (ASR) covered too.

First chapters are dedicated to introduction to VoiceXML language and voice platforms.

Following chapters of this thesis describe design and implementaion of web and voice interface for ticket reservations information system. There is also a description of the configuration of the voice portal included in next chapter.

There were made some benchmark tests of designed voice portal too.

Keywords: VoiceXML, VXML, voice portal, JSON, automatic speech recognition, speech synthesis, JavaScript, Node.js, AngularJS, Asterisk, VoiceGlue, Voxeo Prophecy, SIP, ASR, TTS

Seznam použitých zkratek a symbolů

XML	–	Extensible Markup Language
IP	–	Internet Protocol
TCP/IP	–	Transmission Control Protocol/Internet Protocol
HTTP	–	Hypertext Transfer Protocol
W3C	–	World Wide Web Consortium
VoIP	–	Voice over Internet Protocol
SPA	–	Single Page Application
HTML	–	Hypertext Markup Language
AJAX	–	Asynchronous JavaScript and XML
JSON	–	JavaScript Object Notation
REST	–	Representational State Transfer
API	–	Application Programming Interface
URL	–	Uniform Resource Locator
TCP	–	Transmission Control Protocol
PDF	–	Portable Document Format
ASR	–	Automatic Speech Recognition
TTS	–	Text To Speech
DTMF	–	Dual-Tone Multi-Frequency signaling
IVR	–	Interactive Voice Response
MIME	–	Multipurpose Internet Mail Extensions
SMS	–	Short Message Service
CLI	–	Command Line Interface
AGI	–	Asterisk Gateway Interface
SIP	–	Session Initiation Protocol
CPU	–	Central Processing Unit
CSV	–	Comma-Separated Values
RHEL	–	Red Hat Enterprise Linux
MRCP	–	Media Resource Control Protocol
CCXML	–	Call Control eXtensible Markup Language
SRGS	–	Speech Recognition Grammar Specification
SISR	–	Semantic Interpretation for Speech Recognition

Obsah

1 Úvod **9**

2 VoiceXML **11**
2.1 Architektura 11
2.2 Hlasové platformy VoiceXML 12
2.2.1 Voxeo Prophecy 12
2.2.2 VoiceGlue 13
2.3 VoiceXML dokument 15
2.3.1 Zpracování VoiceXML dokumentu 17
2.4 Základy jazyka VoiceXML 18
2.4.1 Navigace v hlasové aplikaci 18
2.4.2 Získání vstupu od uživatele 19

3 Technologie Text to Speech **23**
3.1 Textová analýza 23
3.2 Fonetická konverze a syntéza řeči 23
3.3 Vybraná TTS řešení 24
3.3.1 Festival Speech Synthesis System 24
3.3.2 Flite 24
3.3.3 Cepstral TTS 25

4 Technologie Automatic Speech Recognition **27**
4.1 Hlasový vstup a jeho numerická reprezentace 27
4.2 Fonetické mapování a generování výstupu 28
4.3 Vybraná ASR řešení 28
4.3.1 Loquendo 28
4.3.2 Laboratoř počítačového zpracování řeči TU Liberec 29

5 Návrh webového portálu **31**
5.1 Uživatelské rozhraní 32
5.2 Použité technologie 35
5.2.1 JavaScript 35
5.2.2 AngularJS 35
5.2.3 Node.js 35
5.2.4 Express.js 37
5.2.5 MongoDB 37
5.2.6 WebSockets 37
5.2.7 Grunt 38
5.2.8 Jade 38

6 Návrh hlasového portálu **39**
6.1 Uživatelské rozhraní . . . 39
6.2 Implementace hlasového portálu . . . 41
6.2.1 Implementace pomocné knihovny . . . 41
6.2.2 Aplikační rozhraní knihovny . . . 44
6.2.3 Popis implementace vybraných částí hlasového portálu . . . 62

7 Konfigurace a zprovoznění hlasového portálu **67**
7.1 Aplikační server . . . 67
7.2 Voxeo Prophecy . . . 68
7.2.1 Instalace . . . 68
7.2.2 Konfigurace . . . 69
7.3 VoiceGlue . . . 69
7.3.1 Instalace . . . 69
7.3.2 Konfigurace . . . 71

8 Zátěžové testy portálu **73**
8.1 Scénář zátěžových testů . . . 73
8.2 Použité nástroje . . . 74
8.2.1 SIPp . . . 76
8.2.2 top . . . 76
8.2.3 Automatizační skript . . . 77
8.3 Výsledky měření . . . 78
8.3.1 Zátěžový test č. 1 . . . 78
8.3.2 Zátěžový test č. 2 . . . 81
8.3.3 Zátěžový test č. 3 . . . 81
8.3.4 Obecné zhodnocení . . . 81

9 Závěr **87**

10 Použité zdroje **89**

Přílohy **90**

A Přílohy **91**
A.1 Obsah CD . . . 91
A.2 Diagramy jednotlivých částí hlasového portálu . . . 91
A.2.1 V textu uvedené diagramy . . . 91
A.2.2 V textu neuvedené diagramy . . . 91

Seznam tabulek

1 Hodnoty sloužící k určení datového typu formulářových prvků 20
2 Implementované metody REST API 33
3 Textové hodnoty pro definici pauzy a přidružených délek pauzy 59
4 Mapování stisků kláves na znaky 64
5 Varianty provedených zátěžových testů 74

Seznam obrázků

1 Architektura VoiceXML 12
2 Architektura VoiceGlue 14
3 Stromová struktura dokumentu VoiceXML 16
4 Proces převodu textu na řeč 24
5 Proces automatického rozpoznávání řeči 28
6 Stránka se seznamem letů a filtrem 34
7 Stránka s formulářem pro editaci letu 36
8 Hlavní struktura hlasového portálu 42
9 Struktura stavu $getLoginData$ ($GetLoginDataFlow$) 43
10 Struktura komponenty pro zadání data 63
11 Struktura komponenty pro zadání textového vstupu 64
12 Struktura komponenty pro zvolení destinace 65
13 Struktura stavu $destinationSelection$ ($DestinationSelectionFlow$) 66
14 Registrace nové VXML aplikace 70
15 Schéma simulovaného hovoru 75
16 Využití CPU při spuštění 5 souběžných hovorů, celkově 5 hovorů 79
17 Využití paměti RAM při spuštění 5 souběžných hovorů, celkově 5 hovorů 80
18 Využití CPU při spuštění 10 souběžných hovorů, celkově 20 hovorů 82
19 Využití paměti RAM při spuštění 10 souběžných hovorů, celkově 20 hovorů 83
20 Využití CPU při spuštění 60 souběžných hovorů, celkově 60 hovorů 84
21 Využití paměti RAM při spuštění 60 souběžných hovorů, celkově 60 hovorů 85
22 Struktura stavu $createNew$ ($CreateNewFlow$) 92
23 Struktura stavu $cancelActive$ ($CancelActiveFlow$) 93
24 Struktura stavu $listActive$ ($ListActiveFlow$) 93
25 Struktura stavu $filterByArrivalDate$ ($FilterByArrivalDateFlow$) 93
26 Struktura stavu $filterByDepartureDate$ ($FilterByDepartureDateFlow$) 93
27 Struktura stavu $reservationMenu$ ($ReservationMenuFlow$) 94
28 Struktura stavu $reservationsContainer$ ($ReservationsContainerFlow$) . . 94

Seznam výpisů zdrojového kódu

1 VoiceXML dokument 16
2 Vytvoření menu pomocí elementu `<menu>` 18
3 Vytvoření formuláře pro získání uživatelské vstupu 20
4 Definice přípustných hodnot uživatelského vstupu pomocí gramatiky . . 21
5 Zdrojový kód ukázkové aplikace 44
6 Vytvoření hlavního objektu aplikace 46
7 Vytvoření instance třídy `Prompt` 50
8 Vytvoření instance třídy `Say` 51
9 Kód vygenerovaný pro stav s hlasovým modelem `Say` 51
10 Kód vygenerovaný pro stav s hlasovým modelem `Exit` 52
11 Vytvoření stavu s hlasovým modelem `Record` 53
12 Kód vygenerovaný pro stav s hlasovým modelem `Record` 54
13 Kód vygenerovaný pro stav s hlasovým modelem `Ask` 55
14 Vytvoření hlasové gramatiky 56
15 Vytvoření kombinované gramatiky `Choices` 58
16 Vytvoření instance třídy `Audio` 58
17 Vytvoření instance třídy `Silence` 59
18 Vytvoření instance třídy `SayAs` 59
19 Použití zástupného objektu `Var` 60
20 Použití zástupného objektu `Var` s funkčním parametrem 61
21 Konfigurace souboru `/etc/asterisk/manager.conf` 71
22 Konfigurace souboru `/etc/asterisk/sip.conf` 72
23 Konfigurace souboru `/etc/asterisk/extensions.conf` 72
24 Zdrojový kód skriptu `start-bench.sh` 77

1 Úvod

Tématem diplomové práce je návrh a implementace webového a hlasového portálu na platformě VoiceXML. Výběr tématu práce byl motivován mým zájmem navrhnout webový informační systém, který bude zároveň možné ovládat i pomocí samoobslužného hlasového kanálu a přinese tak uživatelům flexibilnější možnosti ovládání. V diplomové práci je popsán návrh webového a hlasového rozhraní informačního systému pro rezervaci letenek.

Druhá až čtvrtá kapitola se zabývá teoretickým úvodem do problematiky návrhu hlasových aplikací postavených na platformě VoiceXML.

V druhé kapitole je nejdříve představen jazyk VoiceXML a struktura VoiceXML dokumentů. Kapitola pokračuje popisem vybraných hlasových platforem, na kterých je možné hlasové aplikace provozovat. Náplní závěrečné části kapitoly je seznámení s praktickým návrhem aplikací pomocí jazyka VoiceXML.

Třetí kapitola představuje problematiku technologie TTS, tedy technologie převodu textu na řeč.

Čtvrtá kapitola se zabývá problematikou automatického rozpoznávání řeči pomocí technologie ASR.

Na začátku páté kapitoly je představena navrhovaná aplikace, tedy informační systém pro rezervaci letenek. Zbytek kapitoly je věnován návrhu webového portálu a použitým technologiím.

Na návrh webového portálu plynule navazuje kapitola šestá, která nejdříve představuje vlastnosti hlasového portálu a poté popisuje jeho samotný návrh. Převážná část kapitoly je věnována návrhu knihovny, která slouží k usnadnění návrhu hlasových aplikací postavených na platformě VoiceXML.

Náplní sedmé kapitoly je popis konfigurace vybraných hlasových platforem a zprovoznění navrženého hlasového portálu.

Poslední kapitola se zabývá provedením zátěžových testů hlasového portálu, kde jsou prakticky zjištěny hardwarové a softwarové požadavky na provozní prostředí hlasového portálu.

2 VoiceXML

VoiceXML je značkovací jazyk založený na jazyku XML. Tento jazyk slouží k popisu hlasových dialogů mezi člověkem a počítačem.

Vývoj jazyka začal v roce 1999. V tomto roce založily společnosti Motorola, IBM, AT&T a Lucent organizaci VoiceXML Forum. Tato organizace vznikla za účelem sjednocení proprietárních řešení těchto firem a výsledkem tohoto snažení vznikl standard VoiceXML. V srpnu roku 1999 organizace VoiceXML Forum uvolnila první specifikaci standardu VoiceXML 0.9. Na jaře roku 2000 následovalo zveřejnění verze 1.0 [1].

Po vydání verze 1.0 došlo k předání standardu konsorciu W3C. Konzorcium W3C poté vydalo v březnu 2004 verzi standardu VoiceXML 2.0 a po získání zpětné vazby od partnerů došlo k vydání rozšíření standardu prostřednictvím verze VoiceXML 2.1. Toto rozšíření je zpětně kompatibilní s verzí 2.0.

Organizace VoiceXML Forum po předání standardu VoiceXML konzorciu W3C nezanikla a stále podporuje vývoj standardu.

2.1 Architektura

Architektura VoiceXML je velmi podobná architektuře webových aplikací. U webových aplikací jsou všechny HTML dokumenty uloženy na serveru. Když uživatel potřebuje stáhnout nějaký dokument, tak prostřednictvím svého prohlížeče vyvolá požadavek. Prostřednictvím HTTP protokolu je tento dokument stažen ze serveru a prohlížeč ho může zpracovat. Výsledkem zpracování je zobrazený dokument, se kterým může uživatel pracovat.

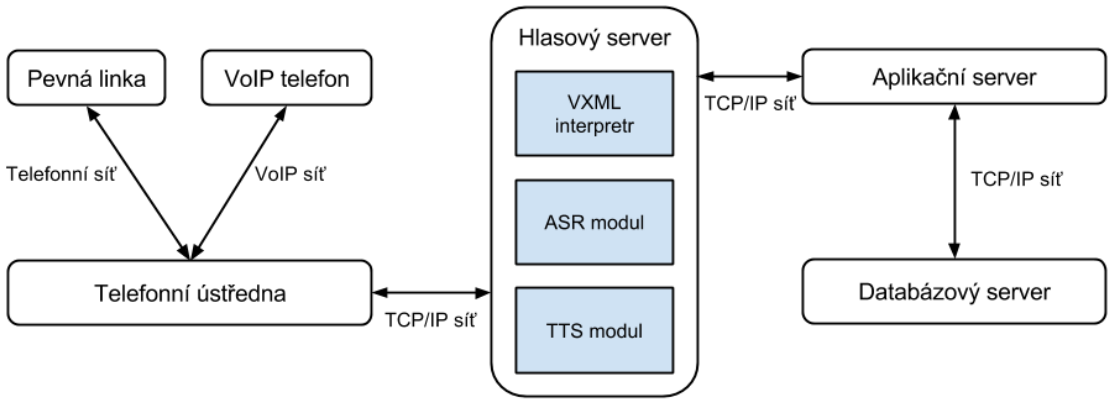
Ve VoiceXML aplikacích jsou dokumenty také uloženy na webovém serveru. Typicky jsou tyto dokumenty generovány dynamicky aplikačním (webovým) serverem. Vedle aplikačního serveru, může být součástí architektury databázový server, který slouží jako zdroj dat.

Mimo aplikační a databázový server, architektura VoiceXML navíc obsahuje hlasový server. Součástí hlasového serveru je interpretr VoiceXML dokumentů, modul pro syntézu řeči (TTS) a modul pro rozpoznávání řeči (ASR). Hlasový server zastává roli prohlížeče, který je používán u webových aplikací. Generuje zvukové výzvy a interpretuje hlasové volby od uživatele.

Uživatel se k hlasovému serveru připojuje prostřednictvím telefonního zařízení. Tímto zařízením může být např. pevná linka, která je součástí veřejné telefonní sítě nebo VoIP telefon, který je připojen do VoIP sítě. Tato zařízení jsou poté prostřednictvím telefonní ústředny připojena k hlasovému serveru.

Jednotlivé servery spolu komunikují prostřednictvím protokolu TCP/IP. Pro přenos dat je typicky zvolen HTTP protokol.

Na obrázku 1 je uvedeno schéma výše popsané architektury [7].



Obrázek 1: Architektura VoiceXML

2.2 Hlasové platformy VoiceXML

V této části představím dvě hlasové (IVR) platformy, nad kterými lze provozovat VoiceXML aplikace. U každé platformy popíši architekturu, uvedu přehled dostupných funkcí a zmíním licenční podmínky, které jsou spojeny s používáním IVR platformy. V neposlední řadě uvedu informaci o podporované verzi jazyka VoiceXML.

Hlasový portál, jehož návrh bude popsán v kapitole 6, bude optimalizován pro provoz právě nad těmito hlasovými platformami.

2.2.1 Voxeo Prophecy

Voxeo Prophecy je komerční IVR platforma. Klíčovou vlastností této platformy je jednoduchá instalace a rychlé uvedení hlasové aplikace do provozu. Součástí distribuce platformy je modul pro syntézu řeči (TTS) a modul sloužící k rozpoznávání řeči (ASR).

Platformu je možné integrovat s veřejnou telefonní sítí. Aplikace provozované nad touto platformou mohou být ovládány pomocí DTMF volby nebo pomocí automatického rozpoznávání řeči. Přestože je součástí platformy JBoss server, nad kterým lze provozovat aplikační server, není tato platforma uzavřena vůči dalším technologiím (např. Node.js, Python).

IVR Voxeo Prophecy je možné provozovat na operačních systémech Windows (32 i 64 bitů), Mac OS X a na linuxových distribucích, které jsou založeny na RHEL (např. CentOS). Pro produkční prostředí je však doporučován operační systém založený na linuxu.

Jak již bylo uvedeno, Voxeo Prophecy je komerční platformou. Provoz této platformy je velmi limitován licenčními podmínkami. Voxeo Prophecy s podporou dvou souběžných hovorů je možné stáhnout zdarma. Aplikaci lze bez registrace provozovat pouze po dobu jednoho měsíce. Můžeme však u Voxea podstoupit bezplatnou registraci a touto registrací odstranit časové omezení. Aplikaci lze poté provozovat po neomezeně dlouhou dobu, omezení na dva souběžné hovory však stále zůstává.

Po registraci máme možnost dokoupit si licenci na provozování více souběžných hovorů (portů) [24]. Pořízení dalších portů však není levnou záležitostí. Na webu společnosti Voxeo je možné najít aktuální ceník.

Platforma Voxeo Prophecy sestává ze čtyř serverů [2], jejichž spuštění se děje automaticky po startu systému:

- Voxeo Prophecy Server
- Voxeo Prophecy VoiceXML Browser
- Voxeo Prophecy Management Console
- Voxeo Prophecy Application Server

Voxeo Prophecy Server je tvořen dvěmi částmi. Jedna část slouží ke zpracování řídících CCXML[1] informací. Druhá část je tvořena MRCP serverem, který slouží ke zpracovávání médií.

Voxeo Prophecy VoiceXML Browser je interpretr, který slouží ke zpracování VoiceXML dokumentů. Tento interpretr plně podporuje specifikaci VoiceXML verze 2.1.

Voxeo Prophecy Management Console představuje rozhraní pro správu hlasových aplikací.

Aplikační server Voxeo Prophecy Application Sever umožňuje provozovat webové aplikace postavené na technologii Java.

Součástí platformy Voxeo Prophecy je dále SIP klient s grafickým rozhraním, kterého lze spouštět přímo ve webovém prohlížeči. Prostřednictvím tohoto klienta je možné, bez další konfigurace, testovat vytvářené aplikace.

Konfigurace hlasových aplikací probíhá prostřednictvím webového rozhraní, jehož jedna stránka je zobrazena na obrázku 14.

Společnost Voxeo je zároveň provozovatelem domény $vxml.org$, kde je možné nalézt dokumentaci k jazyku VoiceXML.

Výhodou této platformy je velmi kvalitní syntetizátor řeči (TTS) a podpora automatického rozpoznávání řeči (ASR). Tyto výhody jsou ale vykoupeny licenčními podmínkami, které nám dovolují na platformě provozovat maximálně dva souběžné hovory.

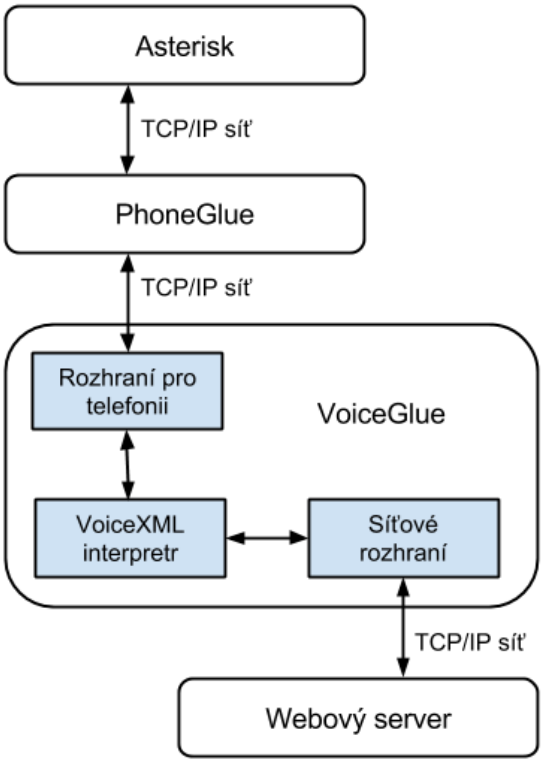
V kapitole 7.2 je uveden podrobný popis instalace a konfigurace platformy Voxeo Prophecy.

2.2.2 VoiceGlue

VoiceGlue samotné netvoří hlasovou platformou. Jedná se o přídavné rozšíření softwarové ústředny Asterisk.

Asterisk je platforma pro provozování telekomunikačních aplikací. Asterisk je projekt s otevřeným zdrojovým kódem, který je uvolněn pod licencí GNU GPL. Vývoj Asterisku je sponzorován společností Digium [3].

[1] CCXML je značkovací jazyk zabezpečující správu, ovládání a přepojování telefonátů.



Obrázek 2: Architektura VoiceGlue

Asterisk ve standardní konfiguraci neposkytuje podporu VoiceXML aplikací, ale díky své modulární architektuře je možné Asterisk o tuto podporu rozšířit.

Asterisk je navržen pro unixové operační systémy (GNU/Linux, OpenBSD, FreeBSD, Mac OS X). K dispozici je i komunitní verze portována do prostředí operačního systému Windows.

VoiceGlue je projekt přinášející podporu VoiceXML aplikací pro platformu Asterisk. VoiceGlue je postaveno nad OpenVXI interpretrem [4]. V době psaní této práce je nejnovější verze 0.14 z listopadu 2011.

VoiceXML interpretr nabízí plnou podporu specifikace VoiceXML 2.0 a zároveň implementuje některé vlastnosti ze specifikace VoiceXML 2.1.

Aplikace provozované na této platformě je možné ovládat pouze pomocí DTMF volby. Součástí VoiceGlue bohužel není modul pro automatické rozpoznávání řeči. VoiceGlue podporuje přehrávání nahraných audio souborů ve formátech, které jsou podporovány Asteriskem, dále podporuje nahrávání vstupu ve formátu $ulaw$.

Flexibilita Asterisku neomezuje provoz VoiceXML pouze na SIP telefonii, ale je možné využít jakýkoliv jiný, Asteriskem podporovaný, telefonní kanál [5].

Do budoucna autoři VoiceGlue slibují integraci s ASR rozpoznávačem LumenVox, kvalitnější TTS syntetizátor a plnou podporu specifikace VoiceXML 2.1.

Součástí VoiceGlue je hlasový syntetizátor Flite (viz kapitola 3.3.2). Flite je možné jednoduchou úpravou konfiguračního souboru VoiceGlue nahradit jiným syntetizátorem.

Podle autorů by nemělo být problém provozovat VoiceGlue na jakémkoliv unixovém operačním systému.

Na obrázku 2 je znázorněna architektura VoiceGlue a propojení s Asteriskem.

Propojení Asterisku s VoiceGlue je úkolem mezivrstvy $PhoneGlue$. Samotnou architekturu VoiceGlue je možné logicky rozdělit na tři hlavní části.

- síťové rozhraní
- část zpracovávající VoiceXML dokumenty
- rozhraní pro telefonii

Síťové rozhraní komunikuje s aplikačním (webovým) serverem, který generuje VoiceXML dokumenty. Komunikace probíhá nad protokolem TCP/IP pomocí protokolu HTTP.

Část zpracovávající VoiceXML dokumenty je tvořena několika dalšími částmi. Jednou částí je OpenVXI interpretr. Další částí je syntaktický analyzátor Xerces. O zpracování případných skriptů jazyka ECMAScript se stará engine SpiderMonkey.

Rozhraní pro telefonii je tvořeno dalšími třemi částmi. První část je připravena pro implementaci rozpoznávání hlasu pomocí ASR rozpoznávače LumenVox (zatím není implementována). Další část tvoří hlasový syntetizátor Flite. Poslední část má na starost komunikaci s mezivrstvou $PhoneGlue$. [2]

Uvedení této hlasové platformy do provozu není tak jednoduché jako v případě platformy Voxeo Prophecy. Nejdříve je potřeba nakonfigurovat Asterisk a poté doinstalovat a nakonfigurovat rozšíření VoiceGlue. Podrobněji je tento proces popsán v kapitole 7.3.

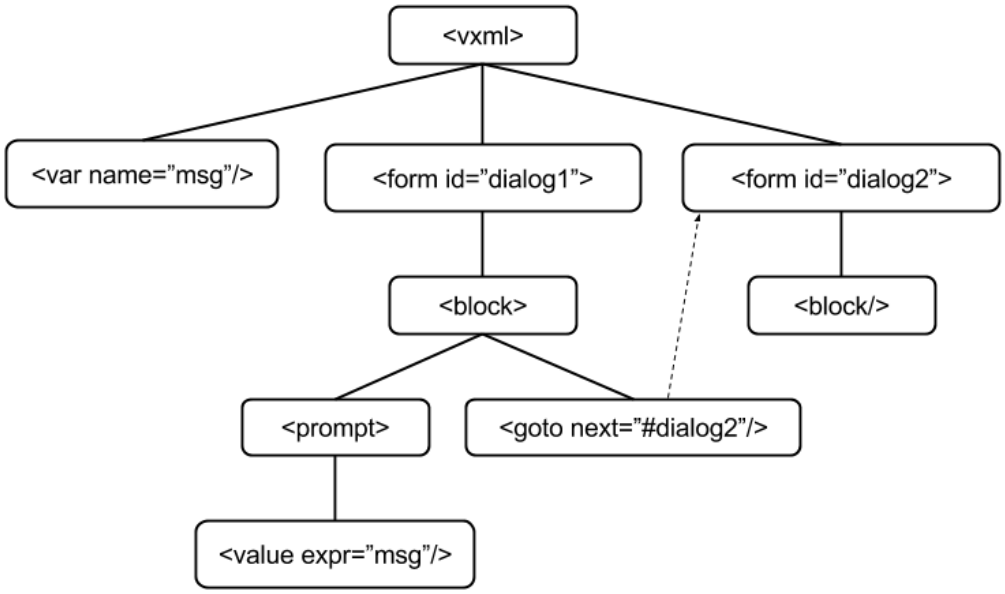
Mezi hlavní výhody VoiceGlue je otevřenost zdrojového kódu, resp. licence pod kterou je VoiceGlue uvolněno. Na druhou stranu má toto řešení zatím dost nevýhod. Jako hlavní nevýhody lze uvést absenci modulu pro rozpoznávání řeči, nekvalitní syntetizátor řeči a chybějící úplnou podporu specifikace VoiceXML verze 2.1.

Vzhledem k tomu, že poslední aktivita autorů VoiceGlue byla na jejich webu zaznamenaná koncem roku 2011, nemůžeme s určitostí v blízké době počítat s implementací těchto chybějících vlastností.

2.3 VoiceXML dokument

Jazyk VoiceXML vychází ze značkovacího jazyka XML a proto pro něj platí veškerá pravidla spojená s jazykem XML. Každý VoiceXML dokument musí splňovat následující pravidla [6]:

- obsahuje právě jeden kořenový element
- neprázdné elementy musí být ohraničeny startovací a ukončovací značkou
- všechny hodnoty atributů musí být uzavřeny v uvozovkách



Obrázek 3: Stromová struktura dokumentu VoiceXML

- elementy mohou být vnořeny, ale nemohou se překrývat

Ve výpise 1 je uvedena ukázka jednoduchého VoiceXML dokumentu. Jednotlivé elementy dokumentu tvoří stromovou strukturu. Struktura tohoto VoiceXML dokumentu je zachycena na obrázku 3.

VoiceXML dokument bývá typicky členěn do několika dialogů a subdialogů. Jednotlivé dialogy obsahují instrukce popisující interakce s uživatelem aplikace. Každý dialog může zároveň obsahovat informaci o tom, do jakého dalšího dialogu se má po dokončení zpracování dialogu pokračovat.

Subdialogy lze chápat jako pomocné funkce, které jsou volány z rodičovského dialogu a mohou provést nějakou akci a poté vrátit výsledek zpět rodičovskému dialogu.

Dialogy můžeme ve VoiceXML rozdělit na dva typy. Prvním typem jsou formuláře. Formuláře popisují interakci vedoucí k získání uživatelského vstupu a pomocí gramatik definují přípustné hodnoty pro vstup. Druhým typem jsou menu. Menu dávají uživateli na výběr ze seznamu voleb a po provedení výběru provedou určenou akci (přesun do jiného dialogu).

```
1 <?xml version="1.0"?>
2 <vxml version="2.0">
3   <var name="msg" expr="'Hello world!'"/>
4
5   <form id="dialog1">
6     <block>
7       <prompt>
```

```
8           <value expr="msg"/>
9         </prompt>
10
11        <goto next="#dialog2"/>
12      </block>
13    </form>
14
15    <form id="dialog2">
16      <block>
17        Second dialog.
18      </block>
19    </form>
20  </vxml>
```

Výpis 1: VoiceXML dokument

2.3.1 Zpracování VoiceXML dokumentu

VoiceXML dokument je zpracováván interpretrem. Zpracování dokumentu začíná u kořenového elementu a postupně se pokračuje k dceřiným elementům.

Nyní se vraťme k výpisu 1 v němž je uveden zdrojový kód jednoduchého VoiceXML dokumentu.

Na prvním řádku můžeme vidět definici verze XML dokumentu. Na dalším řádku specifikujeme verzi VoiceXML dokumentu. Touto informací sdělujeme interpretru verzi standardu VoiceXML, dle které se bude dokument zpracovávat.

Uvnitř elementu `<vxml>` je obsažen kód popisující chování hlasové aplikace. Na třetím řádku definujeme pomocí elementu `<var>` proměnnou a zároveň jí nastavujeme hodnotu.

Následuje definice dvou dialogů. První dialog uživateli přehraje zprávu uloženou v proměnné, kterou jsme definovali na třetím řádku. Následně přechází zpracování dokumentu do dialogu s identifikátorem ”dialog2”.

Tento přechod zajišťuje element `<goto>`. Nastavením hodnoty atributu `next` elementu lze specifikovat identifikátor dialogu, do kterého se má přejít.

Druhý dialog uživateli následně přehraje definovanou zprávu.

Zpracování tohoto dokumentu je možné rozdělit do následujících kroků [1]:

1. voiceXML dokument je nahrán do paměti
2. je provedena kontrola, zda je dokument validním XML dokumentem
3. provede se inicializace proměnných (pokud jsou v dokumentu nějaké definované)
4. provede se načtení prvního dialogu a je spuštěn obsah elementu `<block>`. Zpracováním elementu `<goto>` dochází k přechodu do dalšího dialogu

5. z paměti je uvolněn první dialog, následně dochází k načtení druhého dialogu. Poté jsou provedeny akce definované uvnitř tohoto dialogu

6. druhý dialog je uvolněn z paměti a zpracování VoiceXML je ukončeno

2.4 Základy jazyka VoiceXML

V následující sekci stručně popíši základní postupy, kterých bývá užito při návrhu hlasových aplikací pomocí jazyka VoiceXML.

2.4.1 Navigace v hlasové aplikaci

VoiceXML nám poskytuje několik elementů, pomocí kterých lze jednoduše navrhovat navigační prvky mezi jednotlivými částmi aplikace. My si zde blíže představíme pouze element `<menu>`.

Tento element umožňuje jednoduchým způsobem sestavit interaktivní menu, ve kterém může uživatel provést volbu. Na základě jeho volby dochází k přesunu do jiné části aplikace.

Ve výpise 2 je uveden zdrojový kód [24], který demonstruje vytvoření navigačního menu pomocí elementu `<menu>`.

```
1  <?xml version="1.0"?>
2  <vxml version="2.0">
3    <menu scope="dialog">
4      <prompt>
5        Please select one of the following options:
           ↪ reservations info or account info.
6      </prompt>
7
8      <choice next="#reservations">
9        reservations info
10     </choice>
11     <choice next="#account">
12       account info
13     </choice>
14
15     <nomatch>
16       <reprompt/>
17     </nomatch>
18   </menu>
19
20   <form id="reservations">
21     <block>
22       Reservations info dialog.
```

```
23          <exit/>
24        </block>
25      </form>
26
27      <form id="account">
28        <block>
29          Account info dialog.
30          <exit/>
31        </block>
32      </form>
33  </vxml>
```

Výpis 2: Vytvoření menu pomocí elementu `<menu>`

Jednotlivé položky menu jsou definovány pomocí elementu `<choice>`, přičemž hodnotou atributu `next` specifikujeme do jakého dialogu se má, v případě výběru této položky, přejít.

Můžeme si všimnout, že element `<menu>` dále obsahuje element `<nomatch>`. Tímto elementem je možné specifikovat akci, která se provede v případě, že uživatel provede nedefinovanou volbu. V našem případě je tato situace obsloužena zopakováním položek menu (element `<reprompt>`). Na základě výběru dochází k přesunu do dialogu "reservations" nebo "account", kde je aplikace následně ukončena.

2.4.2 Získání vstupu od uživatele

Jazyk VoiceXML nám pro získání uživatelského vstupu poskytuje podobné prostředky jako jazyk HTML. Máme k dispozici elementy, prostřednictvím kterých jsme schopni nadefinovat formulářové prvky. Do těchto prvků je uložen vstup od uživatele a následně odeslán ke zpracování na server např. prostřednictvím elementu `<submit>`. Elementy pro definici formulářových prvků jsou, stejně jako element `<submit>`, obsaženy uvnitř elementu `<form>`. Zpracováním elementu `<submit>` dochází k odeslání všech formulářových prvků, které jsou ve formuláři obsaženy.

Formulářové prvky se definují pomocí elementu `<field>`. Datový typ formulářového prvku je specifikován prostřednictvím hodnoty atributu `type`. V tabulce 1 je uveden výběr hodnot, které je možné použít [24].

Ve výpise 3 je uveden zdrojový kód aplikace, která vytváří tři formulářové prvky. Tyto formulářové prvky jsou uživatelem vyplněny a následně jsou vyplněné informace odeslány ke zpracování na server. Můžeme si všimnout, že každý formulářový prvek je různého datového typu.

Formulář dále obsahuje element `<filled>`. Tento element specifikuje akci, která se provede v momentě, kdy jsou všechny prvky formuláře vyplněny. Naší akcí je odeslání získaných dat na server.

Odeslání dat zajišťuje element `<submit>`. U tohoto elementu je třeba uvést hodnotu atributů `next` a `namelist`. Hodnotou atributu `next` specifikujeme URL adresu, na

Datový typ	Popis
boolean	Hodnota pravda/nepravda
date	Datum zadaný ve formátu *yyyymmdd* (např. 20140421)
digits	Číselná hodnota (např. 123)
currency	Datový pro načtení měny
phone	Telefonní čísla (závislé na hlasové platformě a systémovém nastavení)

Table 1: Hodnoty sloužící k určení datového typu formulářových prvků

kterou se data odešlou. Hodnotou atributu `namelist` je seznam názvů formulářových prvků, jejichž hodnoty se mají odeslat. Jednotlivé názvy jsou odděleny mezerou.

```
1  <?xml version="1.0"?>
2  <vxml version="2.0">
3    <form>
4      <field name="age" type="number">
5        <prompt>
6          How old are you?
7        </prompt>
8      </field>
9
10     <field name="telephone" type="phone">
11       <prompt>
12         What is your telephone number?
13       </prompt>
14     </field>
15
16     <field name="birthday" type="date">
17       <prompt>
18         When is your birthday?
19       </prompt>
20     </field>
21
22     <filled>
23       <submit next="http://www.mujserver.cz/save-data"
           ↪ namelist="age telephone birthday" />
24     </filled>
25   </form>
26 </vxml>
```

Výpis 3: Vytvoření formuláře pro získání uživatelské vstupu

Hodnoty, které je možné do formulářového prvku zadat, je možné specifikovat ještě jedním způsobem. VoiceXML nám prostřednictvím elementu `<grammar>` poskytuje možnost nadefinovat si gramatiku, která definuje přípustné hodnoty pro vstup.

Z pohledu použité vstupní metody můžeme gramatiky rozdělit na gramatiky, které definují přípustnou DTMF volbu, nebo na gramatiky, které definují přípustné hodnoty pro systém automatického rozpoznávání řeči (ASR). Takovou gramatiku je možné specifikovat pro daný element `<field>` nebo pro celý formulář `<form>`.

Ve standardu VoiceXML 2.0 bývá gramatika nejčastěji popsána XML jazykem. Ve výpise 4 je uveden zdrojový kód dialogu, prostřednictvím kterého je uživatel vyzván k vyřčení oblíbené barvy. V této ukázce je ukázána pouze základní práce s gramatikou.

```
1  <?xml version="1.0"?>
2  <vxml version="2.0">
3    <form>
4      <field name="colour">
5        <prompt>
6          What is your favorite colour?
7        </prompt>
8
9        <grammar>
10         <rule id="root" scope="public">
11           <one-of>
12             <item tag="red">red</item>
13             <item tag="blue">blue</item>
14             <item tag="green">green</item>
15           </one-of>
16         </rule>
17       </grammar>
18
19       <nomatch>
20         Sorry you have selected a wrong colour.
21         <reprompt/>
22       </nomatch>
23
24       <noinput>
25         I did not hear anything.
26         <reprompt/>
27       </noinput>
28     </field>
29
30     <filled>
31       Your favourte color is:
32       <value expr="colour"/>
33     </filled>
```

```
34      </form>
35  </vxml>
```

Výpis 4: Definice přípustných hodnot uživatelského vstupu pomocí gramatiky

3 Technologie Text to Speech

Text to Speech (TTS) je technologie sloužící k převodu textu napsaného běžným jazykem na řeč. TTS modul je základním stavebním kamenem každé hlasové platformy. Tento modul nám totiž umožňuje vytvořit uživatelské rozhraní mezi uživatelem a hlasovou aplikací. Proces převodu textu na řeč je složitý proces, který bude podrobněji popsán v následujícím textu.

Existují dva základní přístupy, jakým způsobem lze umělou řeč generovat [9]. První přístup spočívá ve spojování předem nahraných úseků řeči. Těmito úseky mohou být jednotlivé hlásky a dvojhlásky nebo celá slova. Volba délky nahrávaných úseků ovlivňuje kvalitu reprodukce.

Druhý přístup je založen na simulaci charakteristik lidské řeči a umožňuje tak reprodukovat skutečně umělou řeč.

Proces převodu textu na řeč je v jednoduchosti znázorněn na obrázku 4 a jednotlivé fáze tohoto převodu budou popsány v následujících odstavcích.

3.1 Textová analýza

Textová analýza se dá rozdělit do dvou fází. V první fázi dochází k načtení textového vstupu a převedení do formátu, se kterým je syntetizátor schopen pracovat.

V druhé fázi dochází k normalizaci textu. Aby bylo možné text dále zpracovávat, dochází nejdříve ke konverzi slov na tokeny (tokenizace). Vzniklé tokeny se poté expandují do ortografického tvaru. Dochází tak k expanzi zkratek a čísel na slova. Např. token ”28” je převeden na slovo ”dvacetosm” a zkratka ”č.” je převedena na slovo ”číslo”.

Tato expanze může být v některých situacích závislá na kontextu, ve kterém je token použit. Je proto tedy nutné analyzovat i okolí expandovaného tokenu. Tato situace se dá demonstrovat např. na expanzi zkratky ”den.”. Bez znalosti okolí (kontextu) tokenu nevíme, zda má dojít k expanzi na slovo ”denně” nebo ”denního”. Dalším příkladem může být expanze textu ”1 lev lovil 1 zebru”. První výskyt číslovky 1 chceme převést na slovo ”jeden” a druhý výskyt má být expandován na slovo ”jednu”.

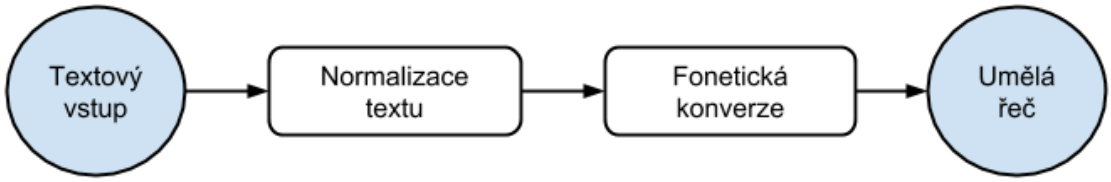
Složitost algoritmů, které tento proces provádějí, je závislá na morfologii[2] interpretovaného jazyka.

3.2 Fonetická konverze a syntéza řeči

V momentě, kdy máme k dispozici normalizovány text, nastává fáze uplatnění fonetických pravidel. V této fázi dochází k převodu každého slova na svůj fonetický ekvivalent [8].

Během tohoto procesu je opět nutné porozumění kontextu, ve kterém se dané slovo vyskytuje. Kontextovou závislost je možné demonstrovat na oznamovací větě ”Honzo, jdeš.” a na tázací větě ”Honzo, jdeš?”. V textové podobě se tyto věty liší pouze znaménkem na konci věty. Pokud jsou ale tyto věty vysloveny, tak se liší v intonaci. A právě intonace dává větám různý význam.

[2]Morfologie je věda, která se zabývá ohýbáním slov, tedy skloňováním slovních druhů a časováním sloves.



Obrázek 4: Proces převodu textu na řeč

Dalším příkladem závislosti fonetické konverze na kontextu může být konverze jednotlivých hlásek. Stejná hláska se nemusí vždy konvertovat na stejnou fonetickou hlásku. U některých jazyků je dokonce žádoucí, aby se daná hláska nepřeváděla na žádnou fonetickou hlásku (neznělé hlásky). Např. ve španělštině se nevyslovuje souhláska "h" na začátku slov.

Posledním krokem je syntéza řeči, která je provedena na základě fonetické konverze.

3.3 Vybraná TTS řešení

Na trhu se vykytuje několik TTS řešení. Některá jsou k dispozici zdarma, v případě jiných se jedná o komerční řešení.

3.3.1 Festival Speech Synthesis System

Festival je obecný vícejazyčný syntetizátor řeči, který byl původně vyvinut Alanem W. Blackem na edinburghské univerzitě. Později se k vývoji připojila i univerzita Canergie Mellon University.

Festival podporuje několik jazyků. Jeho součástí je podpora britské a americké angličtiny, velštiny a španělštiny. Podporu dalších jazyků je možné rozšířit instalací dalších balíčků. Touto formou jsou k dispozici např. následující jazyky: čeština, finština, italština a polština [11].

Program je napsán v programovacím jazyce C++ a je šířen jako svobodný software, je k němu tedy k dispozici i zdrojový kód.

3.3.2 Flite

Název Flite je zkratka ze slovního spojení $festival - lite$. Jedná se o malý a rychlý syntetizátor řeči, který lze provozovat jak na malých (mobilních) zařízeních, tak i na serverových strojích.

Flite je navržen jako odlehčená alternativa k syntetizátoru Festival a zároveň je s tímto syntetizátorem kompatibilní. Program je napsán v programovacím jazyce C a to z důvodu velikosti výsledného souboru a možnosti portace na různé platformy [10].

Za vznikem tohoto syntetizátoru stojí univerzita Carnegie Mellon University. Program je šířen pod licencí X11 a je dostupný zdarma.

VoiceGlue používá Flite jako výchozí syntetizátor.

3.3.3 Cepstral TTS

Společnost Cepstral je poskytovatelem komerčního TTS řešení. Společnost byla založen v roce 2000 vědeckými pracovníky z univerzity Carnegie Mellon University [12].

Cepstral při vývoji svého TTS vyvinul několik nových technik, které umožňují lepší přizpůsobení hlasu dané aplikaci. Cepstral TTS je možné provozovat na zařízeních s omezenými výpočetními zdroji, stejně jako na high-end serverech [13]. Cepstral TTS je možné integrovat se softwarovou ústřednou Asterisk (viz kapitola 2.2.2) a je možné ho provozovat na operačních systémech Windows i na Linuxu.

TTS řešení od Cepstralu nabízí několik různých hlasů, které je možné dále konfigurovat. Jednotlivé hlasy je možné si vyzkoušet na jejich webových stránkách. Cepstral podporuje několik jazyků, konkrétně americkou a britskou angličtinu, americkou španělštinu, kanadskou francouzštinu, němčinu a italštinu.

Přímo na webových stránkách společnosti je možné objednat si jednotlivé hlasy nebo kompletní řešení Cepstral Telephony Server. Cena jednotlivých hlasů pro osobní využití se pohybuje zhruba v rozmezí 10-50 dolarů. Cena Cepstral Telephony Serveru se odvíjí od počtu povolených portů. Základní verze s podporou jednoho portu začíná na 100 dolarech. Za každý další port je nutné si připlatit dalších 100 dolarů.

4 Technologie Automatic Speech Recognition

Technologie ASR, neboli technologie automatického rozpoznávání řeči, je další důležitou součástí hlasové platformy. Uživateli umožňuje ovládat hlasovou aplikaci pomocí hlasových pokynů.

Dnešní systémy pro automatické rozpoznávání řeči dosahují přesnosti rozpoznávání, která se může pohybovat až kolem 70% [1]. K dispozici jsou i systémy, které dokáží rozpoznávat i komplexní jazyky mající širokou slovní zásobu a složitou strukturu, např. čínštinu nebo češtinu.

ASR systémy můžeme rozdělit podle toho, zda jsou orientovány na rozpoznávání souvislé řeči nebo na rozpoznávání izolovaných slov. Rozpoznávání souvislé řeči je, ve srovnání s rozpoznáváním izolovaných slov, mnohem náročnější proces. Při rozpoznávání souvislé řeči totiž dochází k nežádoucím jevům jako je např. spodoba[3] slov nebo polykání hlásek. Při rozpoznávání izolovaných slov k těmto jevům nedochází a je proto rozpoznávání výpočetně i algoritmicky jednodušší.

Pro automatické rozpoznávání řeči se používá několik metod. První metodou je použití algoritmu skrytého Markovova modelu. Jedná se o statistický model, který obsahuje skryté stavy. V kombinaci s tímto algoritmem bývá použit Viterbiho algoritmus za účelem nalezení nejpravděpodobnější posloupnosti skrytých stavů. Tato metoda je vhodná pro rozpoznávání plynulé řeči [14].

Další používanou metodou je metoda dynamického borcení času. Tato metoda je využívána pro rozpoznávání izolovaných slov. Podstatou této metody je měření podobnosti mezi dvěma nahrávkami slov a snaha najít nejlepší shodu.

ASR systémy můžeme rozdělit i podle závislosti, resp. nezávislosti na mluvčím. Systémy, které jsou závislé na konkrétním mluvčím dosahují pro tuto osobu lepších výsledků. Nevýhodou tohoto typu systému je, že daná osoba musí, pro účely natrénování použitelného modelu, namluvit několik hodin nahrávek.

Univerzálnějším typem ASR systému jsou systémy nezávislé na mluvčím. V tomto případě se model sestavuje z nahrávek, které jsou pořízený různými osobami. Nevýhodou těchto systémů je nižší přesnost rozpoznání. Je proto nutné použít další metody, které slouží k adaptaci na konkrétního mluvčího.

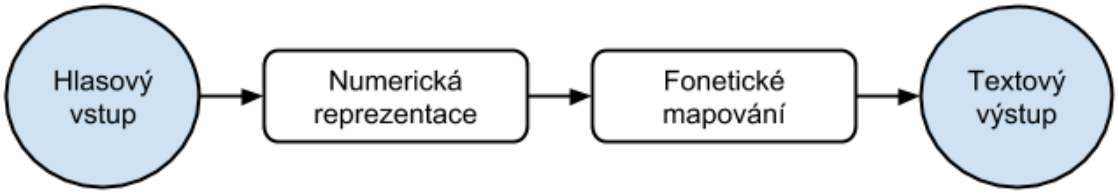
Na obrázku 5 je zachyceno schéma procesu automatického rozpoznávání řeči.

4.1 Hlasový vstup a jeho numerická reprezentace

První fází procesu rozpoznávání řeči je digitalizace řeči, tedy zachycení uživatelského vstupu (řeči) a jeho převod do digitální podoby (nahrávky).

Druhou fází je transformace digitálních dat do numerické modelu. Numerický model zachycuje informace o výšce hlasu v jednotlivých částech nahrávky a další důležité informace potřebné pro další zpracování.

[3] Poslední hláska ve slově je ovlivněna následující hláskou, kterou začíná druhé slovo.



Obrázek 5: Proces automatického rozpoznávání řeči

4.2 Fonetické mapování a generování výstupu

Třetí fází procesu rozpoznávání řeči je fonetické mapování. Tato fáze je nejvíce komplexní a od její přesnosti se odvíjí přesnost celého systému. V této fázi je provedena spektrální analýza numerického modelu. Na základě výsledku spektrální analýzy jsou vybrány pravděpodobnostně nejvhodnější fonémy[4] [1].

Výsledkem fonetického mapování je vygenerování sady řetězců, které vznikly poskládáním z jednotlivých fonémů.

Každému řetězci je přiřazeno skóre, které vyjadřuje podobnost k právě rozpoznávanému slovu. Na základě této podobnosti a gramatiky, která popisuje hodnoty přípustné pro vstup, dochází k vybrání nejvhodnější řetězce. Tento řetězec je poté poslán na výstup.

4.3 Vybraná ASR řešení

Na trhu je k dispozici několik ASR řešení, která jsou převážně komerčního charakteru. Najdeme mezi nimi ale i řešení, která vznikla na půdě univerzit a mohou být k dispozici zdarma. Taková řešení bohužel většinou nedosahují takových kvalit jako řešení komerční.

4.3.1 Loquendo

Loquendo je ASR systém od společnosti Nuance, který je možné provozovat na zařízeních od kapesních zařízení po desktopové počítače. Jedná se o systém, který je nezávislý na mluvčím a umožňuje rozpoznávání plynulé řeči nebo izolovaných slov. Systém si dokáže poradit i s rozpoznáváním řeči v hlučném prostředí [15].

Loquendo dokáže pracovat s velmi širokou slovní zásobou (až 1 milion slov). Systém dosahuje vysoké přesnosti použitím neuronových sítí a algoritmu skrytého Markovova modelu.

Díky své podpoře standardů pro popis gramatik (SRGS 1.0, SISR 1.0) je systém velmi dobře optimalizován pro využití ve VoiceXML aplikacích.

Loquendo podporuje rozpoznávání celkem 28 jazyků, mezi kterými nechybí např. britská a americká angličtina, holandština, dánština, řečtina a francouzština. Loquendo bohužel nepodporuje rozpoznávání češtiny [17].

[4] Foném je nejmenší součást zvukové stránky řeči, která ještě má rozlišovací funkci v systému konkrétního jazyka.

Loquendo je možné si vyzkoušet vytočením telefonního čísla (předvolba USA), kde jsou připraveny tři ukázkové aplikace.

4.3.2 Laboratoř počítačového zpracování řeči TU Liberec

Tato laboratoř patří k vedoucím pracovištím v oboru hlasových technologií v České republice. Začátkem devadesátých se tato laboratoř začala věnovat problematice rozpoznávání izolovaných slov a frází. Později se začala věnovat i rozpoznávání plynulé řeči s orientací na český jazyk.

ASR systémy vyvinuté touto laboratoří zvládají rozpoznávání izolovaných slov z rozsáhlých slovníků, které obsahují stovky tisíc slov. Systém rozpoznávání je nezávislý na mluvčím a v reálném čase dokáže rozpoznávat s úspěšností 95%. ASR systémy lze provozovat i na mobilních zařízeních.

U systémů pro rozpoznávání plynulé řeči se úspěšnost, s použitím slovníku, který obsahuje 350 tisíc slov, pohybuje kolem 90%. V případě přepisu televizních nebo rozhlasových pořadů se úspěšnost pohybuje až kolem 95 - 97% [16].

Laboratoř vyvinula několik aplikací, které využívají schopnosti rozpoznávání řeči. Můžeme uvést např. aplikaci MobilDictate, která slouží k diktování textů do mobilního zařízení nebo aplikaci ATT (Audio Transcription Toolkit), která slouží k automatickému generování titulků pro televizní pořady.

5 Návrh webového portálu

Na začátku této kapitoly nejprve představím referenční aplikaci, na které budu poté ve zbývajícím textu kapitoly demonstrovat proces jejího návrhu a následnou implementaci.

Jako referenční aplikaci jsem zvolil rezervační systém smyšlené letecké společnosti. Rezervační systém obsahuje webové a hlasové rozhraní pro správu rezervací letů. Rezervace lze tedy spravovat z prostředí webového prohlížeče nebo z prostředí samoobslužné hlasové linky.

Pro práci s rezervačním systémem je nutné se do systému nejdříve přihlásit. Každý uživatel se prokazuje kombinací telefonního čísla a hesla.

Do systému se můžou přihlásit dva typy uživatelů. V závislosti na roli přihlášeného uživatele jsou uživateli dovoleny akce, které může vykonávat.

Prvním typem uživatele je uživatel v roli zákazníka. Zákazníkovi je dovoleno vytvoření nové rezervace nebo zrušení již aktivní rezervace, dále je zákazníkovi umožněno vyhledávání letů dle následujících kritérií:

- místa a data odletu
- místa a data příletu
- ceny letu
- maximálního počtu přestupů
- maximální délky letu

Druhým typem uživatele je uživatel v roli administrátora. Pro uživatele s právy administrátora jsou dostupné stejné akce jako uživateli v roli zákazníka, navíc však administrátor může vykonávat následující akce:

- správa databáze letů (editace/vytvoření)
- správa uživatelů (editace/vytvoření)
- správa přepravních společností (editace/vytvoření)

Webové rozhraní rezervačního systému je v reálném čase synchronizováno s hlasovým rozhraním pomocí technologie WebSockets[5] Toto v praxi znamená následující. Pokud je uživatel přihlášen do webového rozhraní a zároveň se nachází v interakci s hlasovou samoobsluhou, kde např. zruší svoji rezervaci, okamžitě dojde k reflektování provedené akce do webového rozhraní, aniž by uživatel musel znovu provést načtení zobrazené stránky.

Webové rozhraní rezervačního systému nabízí uživateli pohodlné grafické rozhraní pro správu rezervací. Webová aplikace je navržena jako jednostránková aplikace[6]. Tento

[5]Webová technologie umožňující vytvoření obousměrného komunikačního kanálu mezi klientem a serverem nad HTTP protokolem. Více informací o této technologii najdete v kapitole 5.2.

[6]Moderní způsob návrhu webové aplikace, kdy je aplikace reprezentována jednou základní HTML stránkou (statickou) a veškeré další interakce na stránce jsou řešeny pomocí JavaScriptu.

způsob návrhu aplikace uživateli přináší rychlejší odezvu uživatelského rozhraní a zároveň minimalizuje množství přenesených dat po síti mezi klientem a serverem. Úspory přenesených dat se dosahuje využitím AJAXového načítání dat, kdy dochází k načtení pouze těch dat, která jsou aktuálně potřebná.

V aplikaci je implementované REST API[7], které slouží jako datový zdroj. REST API implementuje přístup k datům, která jsou uložena v databázi. Většina metod REST API generuje notifikace pro ostatní připojené klienty. Notifikace slouží k synchronizaci uživatelských rozhraní všech připojených klientů.

Data se mezi klientem a serverovou částí aplikace (RESP API) přenáší ve formátu JSON. Formát JSON preferuji hlavně z toho důvodu, že pro zakódování informace je potřeba menšího objemu dat než např. ve srovnání s formátem XML.

Implementace REST API však volitelně dovoluje použít pro komunikaci např. i formát XML. Stačí jen o tom do HTTP požadavku přidat informaci.

Možnost zvolit si formát přenášených dat je umožněna díky malé knihovně, která vznikla zároveň s touto prací. Tato knihovna řeší univerzálním způsobem převod jakéhokoliv JSON objektu do XML formátu.

V tabulce 2 se nachází popis implementovaných metod REST API. API běží na základní URL adrese `/api/v1/`[8] a to z toho důvodu, aby bylo možné jednotlivé verze API v budoucnosti případně verzovat.

5.1 Uživatelské rozhraní

Uživatelské rozhraní aplikace je rozděleno do několika stránek. Po přihlášení do aplikace je zobrazena hlavní stránka se seznamem všech letů (viz obr. 6). Jednotlivé lety jsou zde zobrazeny v seznamu, který je stránkovatelný a je zde možnost nastavení počtu zobrazených položek na stránce. Každý řádek v seznamu reprezentuje jeden let. U každého letu jsou uvedeny základní informace. Na konci každého řádku se nachází tlačítka sloužící pro vytvoření nebo zrušení rezervace letu. Řádek, který představuje rezervovaný let je graficky odlišen jinou barvou pozadí. Dále je zde k dispozici tlačítko pro zobrazení detailních informací o letu. Pokud je přihlášen uživatel s právy administrátora, tak má možnost informace o letu i upravovat. Poslední tlačítko slouží ke smazání letu z databáze. Toto tlačítko se opět zobrazuje jen v případě, že je přihlášen uživatel s právy administrátora.

Nad seznamem letů se nachází formulář pro nastavení vyhledávacích kritérií, dle kterých lze poté seznam letů vyfiltrovat.

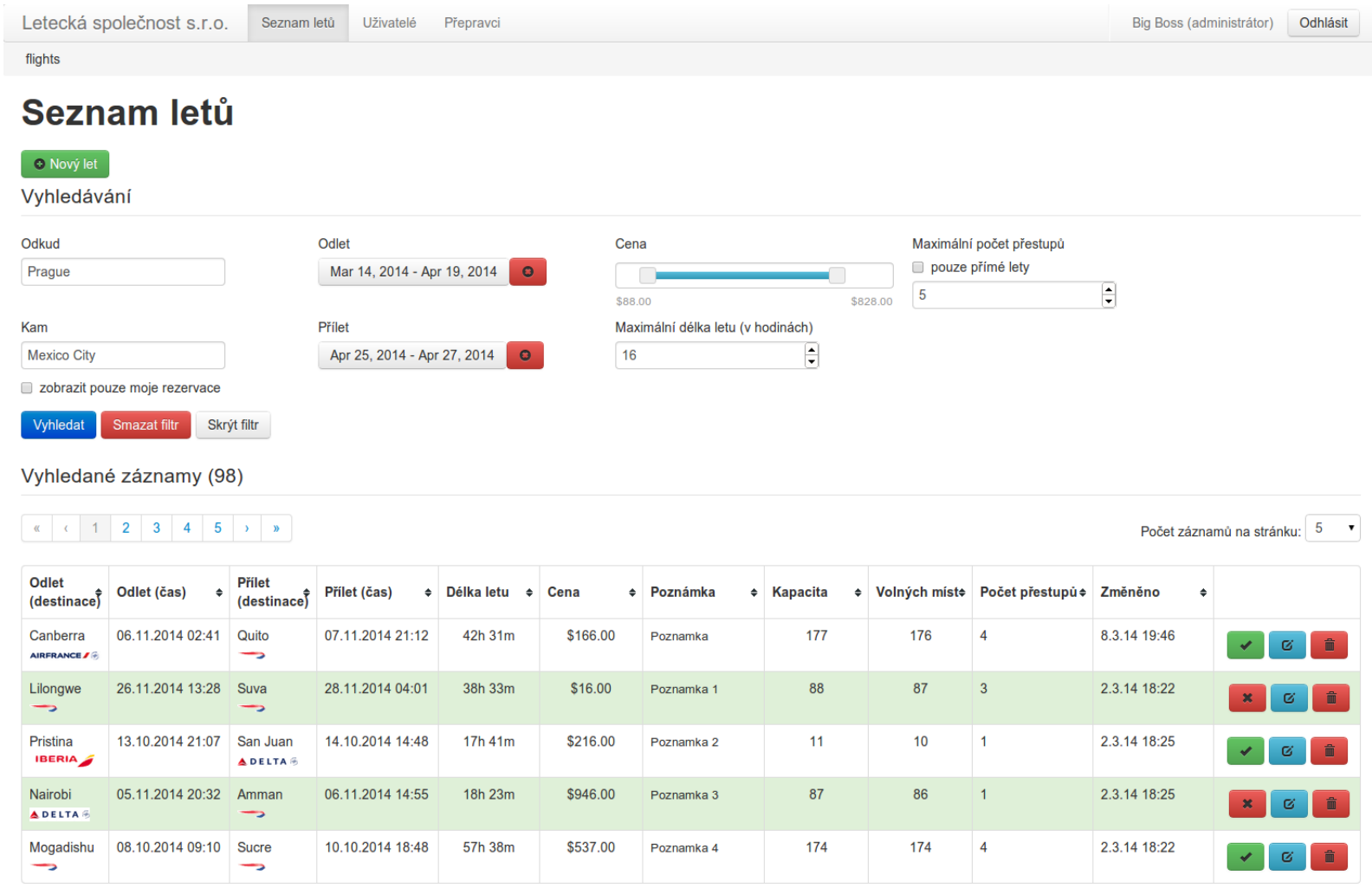
V horní části stránky se nachází menu, které je viditelné ze všech stránek webového rozhraní. Menu obsahuje odkazy na další stránky aplikace jako je stránka se seznamem uživatelů a stránka se seznamem přepravců. V pravé části horního menu se nachází informace se jménem aktuálně přihlášeného uživatele a tlačítko, které slouží k odhlášení uživatele.

[7] REST API je architektura rozhraní navržená pro distribuované prostředí. REST je, na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně.

[8] URL zdroje pro získání informací o jednom letu tedy vypadá následovně `http://muj.server.cz/api/v1/fligths/1`.

HTTP metoda	URL adresa	Popis metody
GET	/flights	Vrací stránkovaný seznam (vyfiltrovaných) letů.
GET	/flights/{id}	Vrací informace o letu.
GET	/flights/{id}/make-reservation	Provádí rezervaci letu pro právě přihlášeného uživatele.
GET	/flights/{id}/cancel-reservation	Ruší rezervaci letu pro právě přihlášeného uživatele.
POST	/flights	Zakládá nový let.
DELETE	/flights/{id}	Odstraňuje let.
PUT	/flights/{id}	Aktualizuje informace o letu.
GET	/users	Vrací stránkovaný seznam uživatelů.
GET	/users/{id}	Vrací informace o uživateli.
GET	/users/{id}/list-reservations	Vrací seznam rezervací pro daného uživatele.
GET	/users/check-login?login={login}	Zjišťuje, zda dané uživatelské jméno již existuje.
POST	/users	Zakládá nového uživatele.
DELETE	/users/{id}	Odstraňuje uživatele.
PUT	/users/{id}	Aktualizuje informace o uživateli.
GET	/carriers	Vrací stránkovaný seznam přepravců.
GET	/carriers/{id}	Vrací informaci o přepravci.
POST	/carriers	Zakládá nového přepravce.
DELETE	/carriers/{id}	Odstraňuje přepravce.
PUT	/carriers/{id}	Aktualizuje informace o přepravci.
GET	/destinations?q={destination}	Vrací seznam vyfiltrovaných destinací. Metoda slouží pro našeptávač.
POST	/security/login	Přihlašuje uživatele.
POST	/security/logout	Odhlašuje uživatele.
GET	/security/current-user	Vrací data o aktuálně přihlášeném uživateli.

Table 2: Implementované metody REST API



Obrázek 6: Stránka se seznamem letů a filtrem

Ostatní stránky aplikace mají uživatelské rozhraní uspořádané obdobně. Odkazy z horního menu vždy vedou na stránky se seznamovým zobrazením dané entity (lety, uživatelé, přepravci). Z toho seznamu existuje možnost přechodu na detail konkrétní entity. V závislosti na právech přihlášeného uživatele je zde možné entitu případně editovat. Pro ukázku uvádím ještě obrázek zobrazující formulář pro editaci letu7.

5.2 Použité technologie

Klientská i serverová část aplikace je napsána v JavaScriptu. V následujících odstavcích stručně popíši použité technologie a důvod jejich použití.

5.2.1 JavaScript

Je multiparadigmatický skriptovací jazyk ovlivněný hlavně jazyky Java, Perl a Smalltalk. Jedná se o událostmi řízený programovací jazyk.

Od klasických programovacích jazyků se liší hlavně podporou prototypové dědičnosti a skutečností, že vše je objekt (dokonce i funkce). Tento jazyk byl vytvořen Brendanem Eichem v roce 1997. Primárním důvodem vzniku tohoto jazyka byla potřeba možnosti obohacení statických webových stránek. Jazyk prošel několika verzemi, kdy do něj byly postupně přidávány nové vlastnosti. V době psaní této práce (2014), je nejvíce rozšířená verze ECMAScript 5 a postupně se začínají do jazyka implementovat vlastnosti z nadcházející specifikace ECMAScript 6.

Díky platformě Node.js již není JavaScript omezen jen na oblast klientského skriptování, ale pronikl i na stranu serverového programování a umožňuje návrh vysoce škálovatelných síťových aplikací.

5.2.2 AngularJS

AngularJS je framework, který svými unikátními vlastnostmi ulehčuje návrh jednostránkových aplikací v JavaScriptu. Mezi klíčové vlastnosti frameworku patří obousměrný databinding, neboli automatické provázání dat (modelu) s vizuální reprezentací (pohledem). Další užitečnou vlastností Angularu je šablonovací systém, který pomocí direktiv umožňuje rozšířit standardní sadu HTML tagů o nové specifické HTML tagy. Více informací o této technologii najdete v dokumentaci frameworku[18].

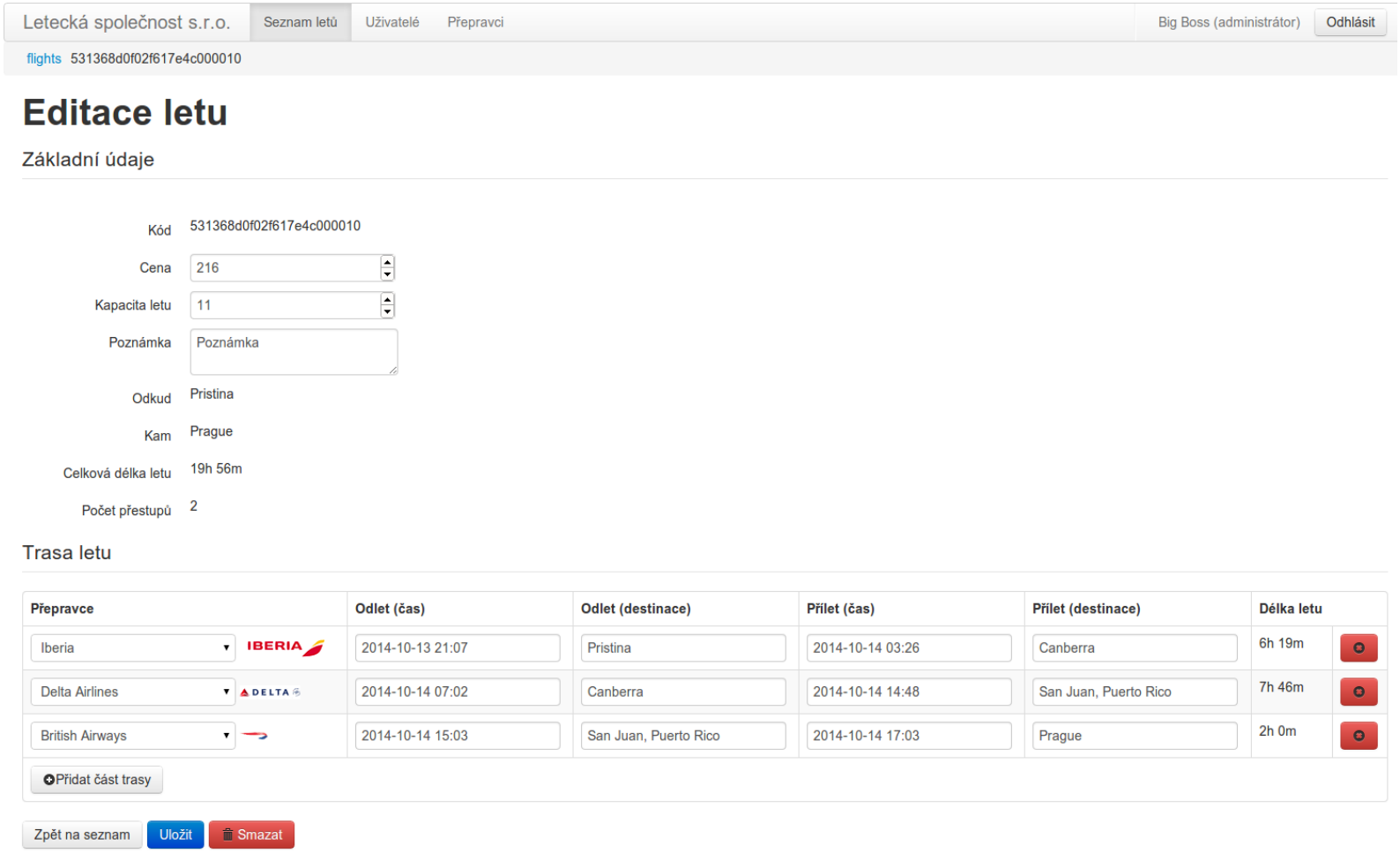
Tento framework jsem použil k návrhu klientské části aplikace a to právě z výše uvedených důvodů.

5.2.3 Node.js

Node.js[21] je technologie přinášející možnost psát serverové části webových aplikací v JavaScriptu.

Platforma Node.js se skládá z běhového prostředí, které je postaveno nad JavaScript enginem V8[9] a z knihovny funkcí. Mezi důležité vlastnosti Node.js patří jeho asynchronní,

[9]Stejný engine je i součástí webového prohlížeče Chromium.



Obrázek 7: Stránka s formulářem pro editaci letu

událostmi řízená povaha, tedy veškeré vstupně/výstupní operace jsou zpracovávány asynchronně. Díky tomuto chování v praxi dochází k tomu, že webový server postavený nad Node.js dokáže zpracovat více souběžných požadavků[10].

Technologii jsem zvolil hlavně proto, že mi umožňuje psát serverovou a klientskou část aplikace ve stejném programovacím jazyce, tedy v JavaScriptu.

5.2.4 Express.js

Jedná se o minimalistický framework pro Node.js, který umožňuje snadné vytváření webových aplikací. Nabízí např. funkce pro práci s cookies, práci s objektem příchozího požadavku a zápis do objektu odpovědi. Framework je jednoduše rozšiřitelný o další funkce.

Více informací o tomto frameworku je možné nalézt na oficiálním webu[22].

5.2.5 MongoDB

Pro ukládání dat jsem zvolil databázi MongoDB. MongoDB je NoSQL databáze, což znamená, že principy přístupu k datům nejsou implementovány na bázi relací jako jsme zvyklí u klasických relačních databází. MongoDB se řadí mezi dokumentové databáze, které nemají pevně definované schéma.

Více informací o MongoDB databázi lze nalézt na webu projektu[19].

V kombinaci s touto databází jsem použil nadstavbu Mongoose[20], což je vrstva přidávající možnost definovat pevné schéma dokumentů a pracovat s těmito dokumenty jako s datovými modely. Tato vrstva dále umožňuje např. přidávání validačních metod na jednotlivé modely.

5.2.6 WebSockets

Je technologie pro tvorbu webových aplikací, které mezi sebou potřebují komunikovat v reálném čase. Technologie je součástí standardu HTML5 a je již v moderních prohlížečích plně implementována. Technologie umožňuje vybudovat plně duplexní komunikační kanál mezi klientem serverem a tímto kanálem si vzájemně vyměňovat data. Jedná se v podstatě o obdobu klasického TCP spojení přeneseného do oblasti webových aplikací. WebSockety komunikují nad standardním HTTP protokolem, nemají proto problém s jakýmikoliv proxy servery a firewally na trase mezi klientem a serverem.

Tuto technologii v aplikaci používám k synchronizaci všech aktuálně otevřených webových rozhraní, aby přihlášený uživatel nemusel otevřenou stránku ve webovém prohlížeči manuálně znovu načítat.

[10] Za předpokladu, že je aplikace orientovaná na vstupně/výstupní operace jako je např. čtení z databáze nebo ze souborového systému.

5.2.7 Grunt

Grunt je automatizační nástroj, který usnadňuje workflow vývoje (nejen) webových aplikací. Tento nástroj nabízí aplikační rozhraní pro definici úkolů, které se mají automaticky provést v definovaný moment. Definice úkolů se popisuje v jazyce JavaScript.

Já jsem tento nástroj použil např. k validaci zdrojového kódu napsaného v JavaScriptu nebo ke spouštění webového serveru. Tento nástroj jsem dokonce využil k automatizaci sazby této práce - po každém uložení zdrojového LaTeX souboru dojde k automatickému vytvoření výstupního PDF souboru.

Více informaci o tomto nástroji je možné nalézt na oficiálních webových stránkách[23].

5.2.8 Jade

Jedná se o šablonovací jazyk používaný na platformě Node.js. Jade svými vlastnostmi urychluje zápis HTML kódu. Např. není potřeba psát ukončovací značky HTML elementů a není potřeba značky zapisovat v úhlových závorkách. Zanoření jednotlivých elementů je vyjádřeno postupným odsazováním elementů. Šablonovací systém je rozšiřitelný pomocí mixinů.

6 Návrh hlasového portálu

V této části práce se zaměřím na popis procesu návrhu a následné implementace hlasového portálu rezervačního systému. Rezervační systém byl podrobněji představen za začátku kapitoly 5.

Hlasový portál umožňuje uživateli spravovat své rezervace prostřednictvím hlasové samoobsluhy, se kterou se spojí vytočením telefonního čísla např. na svém mobilním telefonu. Hlasový portál se od portálu webového liší hlavně ve způsobu interakce uživatele se systémem a množinou dostupných akcí, které jsou uživateli umožněny.

Stručně řečeno je uživateli nabídnuta podmnožina akcí, které jsou dostupné ve webovém portálu. Hlasový portál nabízí užší množinu funkcí hlavně z toho důvodu, že ne všechny funkce systému lze z pohledu uživatelského rozhraní komfortně implementovat i v prostředí hlasového systému. Hlasový portál je tedy cílen pouze na uživatele v roli zákazníka letecké společnosti a nenabízí funkce uživatelům v roli administrátora systému, které jim jsou dostupné prostřednictvím webového portálu (např. přidání nového letu do databáze).

V následujícím seznamu uvádím souhrn akcí, které může uživatel prostřednictvím hlasového portálu vykonávat:

- přihlášení do systému
- založení nové rezervace
- vylistování seznamu aktivních rezervací
- zrušení všech aktivních rezervací
- odhlášení ze systému

V následujícím textu nejdříve popíši odlišnosti v návrhu uživatelského rozhraní hlasového portálu ve srovnání s návrhem uživatelského rozhraní webového portálu a poté se zaměřím na popis návrhu hlasového rozhraní rezervačního systému.

6.1 Uživatelské rozhraní

Uživatelské rozhraní sloužící ke komunikaci mezi uživatelem a hlasovým portálem je diametrálně odlišné od rozhraní, které nabízí webové portály. Proto je nutné při návrhu uživatelského rozhraní hlasového portálu k těmto odlišnostem přihlédnout. Hlavní rozdíl mezi uživatelským rozhraním webových a hlasových portálů je ve způsobu interakce. Zatímco webové portály nabízí uživatelská rozhraní založená převážně na interakci pomocí vizuálních vjemů uživatele, při návrhu uživatelského rozhraní hlasového portálu není možné interakci založenou na vizuálních vjemech aplikovat. Uživatelské rozhraní hlasových portálů je totiž založeno pouze na zvukových vjemech a hlasových odezvách. Proto není vhodné navrhovat příliš komplexní uživatelské rozhraní. Naopak, musíme se snažit uživatelské rozhraní členit co nejjednodušeji a přehledně tak, abychom uživateli

interakci se systémem co nejvíce zjednodušili a nedocházelo tak k dezorientaci v prostředí systému.

Komunikace s uživatelem většinou probíhá pomocí automatické syntézy řeči. Zatím však neexistuje hlasový syntetizátor, který by se svou kvalitou zcela vyrovnal lidské řeči. V závislosti na kvalitě použitého syntetizéru může někdy při syntéze řeči docházet ke komolení slov a pro uživatele nemusí být sdělení zcela srozumitelné. V tomto případě většinou stačí nahradit zkomolenou část věty jiným slovem či slovním spojením. V případě použití nekvalitního syntetizéru řeči se, před nasazením hlasového portálu do produkčního prostředí, doporučuje veškerá hlasová sdělení namluvit a zdigitalizovat a tyto nahrávky používat místo automatické syntézy.

Před začátkem návrhu uživatelského rozhraní hlasového systému je vhodné nejdříve provést analýzu cílové skupiny uživatelů, která bude daný systém používat. Uživatelé z různých cílových skupin mají ve většině případů odlišné požadavky na podobu uživatelského rozhraní. Pokud se například zaměřujeme na skupinu uživatelů staršího věku, můžeme předpokládat, že tito uživatelé nemají žádné nebo velmi omezené zkušenosti s ovládáním hlasového portálu. Proto bychom se měli snažit ovládání systému ještě více zjednodušit. Takového zjednodušení lze dosáhnout například nabídnutím možnosti vyvolání okamžité nápovědy během právě probíhající hlasového interakce. Pro takového uživatele bude zároveň jistě pohodlnější komunikovat se systémem za pomoci automatického rozpoznávání řeči než pomocí DTMF volby.

Naopak pokud víme, že naši aplikaci budou využívat převážně uživatelé, kteří jsou znalí moderních technologií, můžeme si dovolit navrhnout složitější uživatelské rozhraní založené na ovládání pomocí DTMF volby.

Jedno je ale společné pro všechny typy uživatelů. Za každých okolností musíme uživatelům umožnit rychlou a pohodlnou navigaci napříč celou aplikací tak, aby byli schopni se v co nejkratším čase dostat k informaci, kterou hledají.

Já jsem se při návrhu hlasového portálu zaměřil na cílovou skupinu uživatelů, u kterých již předpokládám určitou znalost komunikace s hlasovým portálem[11]. Pro navigaci napříč hlasovým portálem se využívá ovládání pomocí DTMF volby. Ovládání pomocí DTMF volby jsem zvolil hlavně ze dvou důvodů.

Prvním důvodem je potřeba získávat od uživatele přesné vstupy jako je např. zadání data nebo identifikátoru rezervace letu. Tyto vstupy se dají jednoduše zadávat pomocí DTMF volby.

Dalším důvodem zvolení DTMF je široká podpora tohoto typu vstupu ze strany IVR platforem. Mým cílem bylo zprovoznit aplikaci na dvou různých IVR platformách, přičemž jedna z nich nenabízí podporu vstupu pomocí automatického rozpoznávání řeči. V rámci zachování jednotné metody vstupu napříč všemi testovanými IVR platformami jsem tedy zvolil DTMF volbu.

Aby bylo možné s hlasovým rozhraním rezervačního systému začít pracovat, je nutné se do něj nejdříve přihlásit. Každý uživatel se prokazuje svým telefonním číslem a číselným heslem. Pokud uživatel zadá špatné telefonní číslo a heslo, tak je mu nabídnuto

[11] Předpokládám tedy, že uživatelé mají zkušenost alespoň s hlasovou samoobsluhou svého mobilního operátora

opakované zadání. V případě, že se uživatel prokáže správným telefonním číslem, ale špatným heslem a toto zadání provede třikrát za sebou špatně, tak dojde k dočasnému zablokování jeho účtu. V tomto případě je uživateli sdělena informace o tom, kdy se může zkusit znovu přihlásit.

V případě úspěšného přihlášení do systému, je uživateli přehrána uvítací zpráva a následně je uživatel přesměrován do hlavního menu aplikace, kde má k dispozici akce pro správu a vytváření rezervací (seznam akcí je uveden v textu kapitoly 6). Jednotlivá menu aplikace jsou navržena tak, aby měl uživatel kdykoliv možnost vrátit se o úroveň zpět.

Na obrázku 8 je zobrazena hlavní struktura hlasového portálu. Struktura aplikace je na obrázku popsána formou stavového diagramu. Obdélníky reprezentují jednotlivé stavy aplikace. Některé stavy jsou pro svoji komplexnost zobrazeny pouze obecně. U takových stavů je vždy uvedena informace, která slouží jako jakýsi odkaz na další diagram, který daný stav popisuje detailněji. Jako příklad můžu uvést stav pojmenovaný $getLoginData$. U tohoto stavu je ve dvojitých úhlových závorkách uvedena informace $<<$ $GetLoginDataFlow >>$. Tato informace nám říká, že stav $getLoginData$ obsahuje další vnořené stavy. Tyto stavy jsou v aplikaci implementovány třídou $GetLoginDataFlow$. K této třídě existuje diagram, který popisuje vnitřní uspořádání těchto stavů a přechody mezi nimi.

Pro ukázku uvádím ještě diagram této třídy na obrázku 9.

Pomocí šipek jsou v diagramu popsány přechody mezi jednotlivými stavy. U každého přechodu se dále nachází informace identifikující daný přechod. Součástí této informace může být i doplňující informace o DTMF volbě, která slouží jako parametr tohoto přechodu a bývá uvedena v hranatých závorkách.

Z důvodu zpřehlednění textu zde neuvádím diagramy všech částí aplikace. Nejvíce zajímavým částem aplikace se budu detailněji věnovat později. Diagramy ostatních částí aplikace jsou obsaženy v příloze A.2.

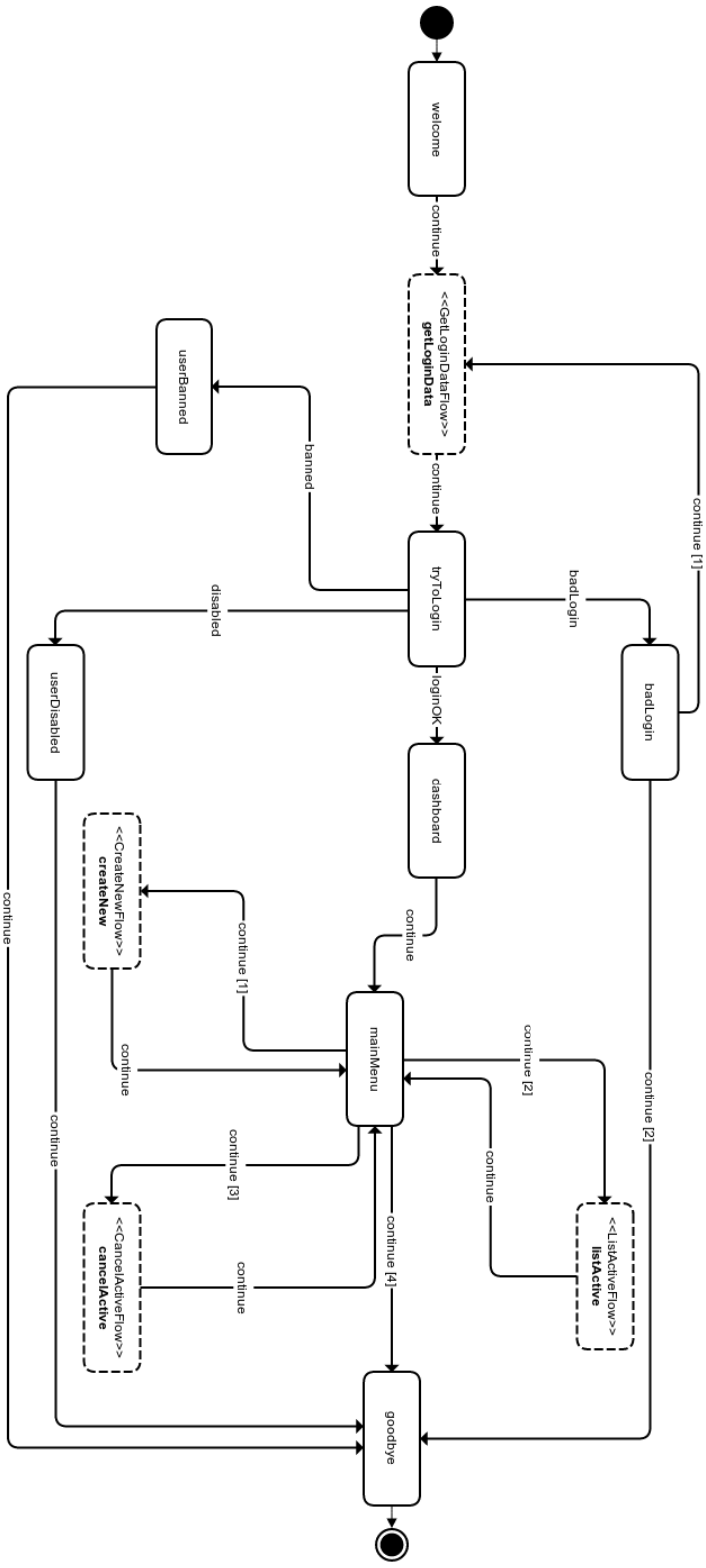
6.2 Implementace hlasového portálu

V této kapitole se budu věnovat popisu implementace hlasového portálu. Nejdříve se zaměřím na popis knihovny, která vznikla v rámci této práce za účelem usnadnění vývoje hlasových portálů postavených na platformě VoiceXML. Poté se vrátím k detailnějšímu popisu implementace vybraných částí hlasového portálu.

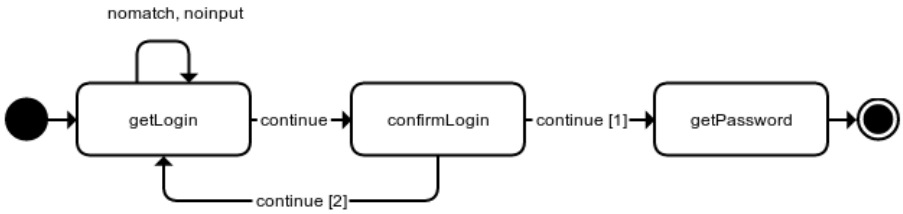
Proces implementace hlasového portálu můžu tedy rozdělit na dvě části. První částí je návrh VXML knihovny a druhou částí je návrh hlasového portálu rezervačního systému s využitím vytvořené knihovny.

6.2.1 Implementace pomocné knihovny

V kapitole 2.3 jsme se seznámili se základy návrhu VoiceXML dokumentů, konkrétně jsme se zde zaměřili na strukturu VXML dokumentu a popis vybraných elementů, které slouží k definici struktury aplikace. Již tedy máme k dispozici znalosti, které nám umožňují napsat VoiceXML kód jednoduchého hlasového portálu.



Obrázek 8: Hlavní struktura hlasového portálu



Obrázek 9: Struktura stavu $getLoginData$ $(GetLoginDataFlow)$

Při návrhu složitějšího hlasového portálu by nám mohlo ušetřit hodně práce vyšší úroveň abstrakce. Tato abstrakce by nás mohla odstínit od přímého psaní VXML kódu. A právě tuto abstrakci zajišťuje knihovna, která je vyvinuta v rámci této diplomové práce.

Při použití této knihovny se nemusíme soustředit na psaní validního VXML kódu, ale veškeré naše soustředění můžeme zaměřit na aplikační logiku. Tato knihovna dále umožňuje dosáhnout efektivní znovupoužitelnosti kódu formou komponentního modelu. V neposlední řadě knihovna zjednodušuje vývoj tím způsobem, že v podstatě ruší rozdělení aplikace na klientskou a serverovou část. Při vývoji aplikace tedy nemáme aplikaci striktně rozdělenou na VXML dokumenty, které definují strukturu aplikace a zajišťují vstup a výstup, a serverovou část, která řeší přístup k datům a implementuje aplikační logiku.

Knihovna je implementována v JavaScriptu[12] a aplikace, která je vyvíjena s využitím této knihovny je spouštěna v serverovém prostředí na platformě Node.js.

Aplikace vytvořená pomocí této knihovny se dá jednoduše popsat jako kolekce stavů mezi kterými jsou definovány přechody do stavů dalších (v podstatě se tedy jedná o stavový automat). Každý stav může být určitého typu. Jedním typem je stav, sloužící k získání vstupu od uživatele. Dalším typem je stav, který zajišťuje výstup, tedy sděluje uživateli nějakou informaci. Posledním typem je stav, který nezajišťuje vstup ani výstup, ale implementuje pouze logiku, na základě které dochází mj. k rozhodnutí do jakého dalšího stavu se bude přecházet.

Každý stav může vyvolávat události. Pro každou vyvolanou událost je definován přechod do dalšího stavu. Způsob definice přechodů pomocí událostí zabraňuje vzniku pevných vazeb mezi objekty, které spolu potřebují nějakým způsobem komunikovat. V rámci interní implementace stavu tedy může docházet k vyvolání události, na které mohou závislé objekty nějakým způsobem reagovat.

Každý stav může definovat akci, která se vyvolá v momentě, kdy dojde ke vstupu do tohoto stavu. Stav zároveň může definovat akci, která se provede těsně před přechodem do nového stavu, tedy při opouštění stavu, který tuto akci definoval. U stavů, které slouží k získání vstupu uživatele se obsluha akce při opouštění stavu typicky využívá ke zpracování vstupních dat (např. zápis dat do databáze).

Každý stav se může skládat z dalších (vnořených) stavů. V případě, že stav vnořené stavy obsahuje, tak se mezi stavy přechází následujícím způsobem. Nejdříve se postupně

[12]Více informací o této technologii můžete najít v kapitole 5.2.1.

navštíví vnořené stavy rodičovské stavu. Jakmile dojde k navštívení vnořeného stavu, který je finální, tak se zpracování vrací k přechodům, které má definován rodičovský stav.

Knihovna nám tedy poskytuje rozhraní API umožňující definovat jednotlivé stavy aplikace, přechody mezi těmito stavy a akce, které se mají vyvolávat při přechodech do dalších stavů. Nemusíme se tedy starat o ruční psaní VXML kódu, který je nutný pro zpracování VXML interpretrem. Tento kód generuje knihovna automaticky.

6.2.2 Aplikační rozhraní knihovny

V předchozí kapitole jsem představil pomocnou knihovnu a popsal jsem důvody, které mě vedly k její implementaci.

V této části se budu podrobně věnovat jednotlivým částem rozhraní API, které nám tato knihovna nabízí. Popíši zde jednotlivé třídy, které knihovna obsahuje. Aby bylo možné srovnat vývoj VoiceXML hlasového portálu za pomoci této knihovny s tradičním způsobem vývoje (kdy ručně vytváříme VXML kód), nesmí chybět i ukázky použití jednotlivých částí knihovny.

Ve výpise 5 je uveden zdrojový kód jednoduché aplikace, jejíž zdrojový kód bude v dalším textu předmětem detailnějšího rozboru. Jen pro představu zde ještě uvedu, co daná aplikace dělá. Aplikace představuje jednoduché menu. Nejdříve nás aplikace přivítá uvítací hláškou. Poté dojde k přechodu do menu, kde se dozvídáme, že můžeme pomocí volby DTMF zvolit hodnotu 1, 2 nebo 3. V závislosti na výběru poté přecházíme do stavu, ve kterém jsme informováni o tom, jakou volbu jsme provedli. Pokud zvolíme jinou hodnotu, než je povoleno (např. 4) dojde k přechodu do stavu, kde je nám řečeno, že jsme zvolili nepovolenou volbu.

```
1  var MenuExampleCtrl = CallFlow.extend({
2
3    constructor: function () {
4      // zavolani konstruktoru bazove tridy
5      MenuExampleCtrl.super.call(this);
6    },
7
8    create: function* () {
9      // vytvoreni jednotlivych stavu
10     var gretingState =
11       State.create('greeting', new Say('Welcome to menu
           ↪ example.'));
12
13     var optionOneState =
14       State.create('optionOne', new Exit('Selected option
           ↪ is one.'));
15
16     var optionTwoState =
```

```
17        State.create('optionTwo', new Exit('Selected option
              ↪ is two.'));
18
19      var optionThreeState =
20        State.create('optionThree', new Exit('Selected option
              ↪  is three.'));
21
22      var invalidSelState =
23        State.create('invalidSelection', new Exit('Invalid
              ↪ option.'));
24
25      var menuState =
26        State.create('menu', new Ask({
27          prompt: 'Press one, two or three.',
28          grammar: new BuiltinGrammar({
29            type: 'digits',
30            length: 1
31          })
32        }))
33        .addOnEntryAction(function* () {
34          console.log('You have entered menu state.');
35        })
36        .addOnExitAction(function* () {
37          console.log('You have leaved menu state.');
38        })
39        .addTransition('continue', optionOneState, function (
              ↪ result) {
40          return result == 1;
41        })
42        .addTransition('continue', optionTwoState, function (
              ↪ result) {
43          return result == 2;
44        })
45        .addTransition('continue', optionThreeState, function
              ↪  (result) {
46          return result == 3;
47        })
48        .addTransition('continue', invalidSelState, function
              ↪ (result) {
49          return [1, 2, 3].indexOf(result) == -1;
50        });
51
52      gretingState.addTransition('continue', menuState);
```

```
53
54      // registrace stavu do kontejneru
55      this
56         .addState(gretingState)
57         .addState(menuState)
58         .addState(optionOneState)
59         .addState(optionTwoState)
60         .addState(optionThreeState)
61         .addState(invalidSelState);
62   }
63 });
```

Výpis 5: Zdrojový kód ukázkové aplikace

6.2.2.1 Třída Application Každá aplikace je reprezentována hlavním objektem aplikace, který celou aplikaci zapouzdřuje. Instance této třídy představuje hlavní objekt aplikace. Hlavním úkolem aplikačního objektu je automatické generování VXML dokumentu, který je vždy vygenerován pro aktuální stav aplikace a slouží jako vstup pro VXML interpretr. Interpretr tento dokument zpracuje a výsledkem je dialog mezi uživatelem a hlasovým systémem.

Tato třída implementuje statickou metodu `create`, která přijímá potřebné parametry k vytvoření aplikace. Ve výpise 6 je uvedena ukázka kódu, kterým se vytvoří instance aplikace.

```
1 Application.create({
2   route: '/menu-example',
3   controller: MenuExampleCtrl,
4   config: {
5     io: io
6   },
7   server: appServer
8 });
```

Výpis 6: Vytvoření hlavního objektu aplikace

Metoda `create` přijímá jako parametr jeden objekt, který obsahuje jednotlivá nastavení aplikace. Následuje popis jednotlivých parametrů.

Parametr `route` udává, na jaké URL bude aplikace dostupná.

Parametr `controller` je reference na konstruktor, který se použije k vytvoření hlavního flow[13] aplikace.

[13] V textu používám termín *flow* pro označení instancí třídy `CallFlow` (viz 6.2.2.2), které definují kolekci stavů aplikace.

Dalším parametrem, který není povinný, je parametr `config`. Jedná se o objekt, pomocí kterého můžeme do aplikace protlačit jakákoliv další nastavení. Tyto nastavení jsou poté dostupné celé aplikaci. V příkladu je uvedeno nastavení `io`, prostřednictvím kterého aplikaci předáváme referenci na objekt, který se stará o generování push notifikací pomocí technologie WebSockets[14].

Hodnotou posledního parametru `server` je instance Express.js serveru (viz 5.2.4), nad kterým je VXML aplikace spuštěna. Tato instance je interně využita k navěšení metod, které obsluhují jednotlivé požadavky na přechod mezi stavy aplikace.

Aplikace běží nad HTTP protokolem. Požadavky na přechod mezi stavy jsou tedy klasické HTTP požadavky. První požadavek na VXML aplikaci má následující tvar:

```
http://www.myserver.com/menu-example?session.sessionid=123
```

Následující požadavky jsou ve tvaru následujícím:

```
http://www.myserver.com/menu-example?sessionid=123
&event=continue&result=1
```

V URL adrese požadavku jsou zakódovány všechny potřebné informace pro provedení přechodu do dalšího stavu.

První výše uvedený požadavek obsahuje pouze jeden parametr, a tím je parametr `session.sessionid`. Tento parametr slouží jako jednoznačný identifikátor instance aplikace. Hodnotu tohoto parametru generuje IVR platforma a tuto hodnotu dále předává aplikačnímu serveru. Aplikační server tento údaj využívá jako klíč, pod kterým ukládá stav aplikace mezi jednotlivými požadavky. Informace o stavu, která je na serveru uchovávána, obsahuje například informaci o právě navštíveném stavu.

Identifikátor instance aplikace je poté obsažen ve všech dalších požadavcích prostřednictvím parametru `sessionid`. Mezi dalšími požadavky je hodnota předávána prostřednictvím automaticky vygenerovaného VXML dokumentu. Pokud by hodnota nebyla přítomna, aplikace by nevěděla s jakou instancí má pracovat.

Dalším parametrem, který musí být v URL obsažen, je název události `event`, která byla vyvolána v interakci s uživatelem. Na základě této informace server zpracuje přechod do dalšího stavu.

V URL adrese požadavku může být obsažen ještě jeden parametr. Jedná se o parametr `result`. Pokud se právě zpracovává stav, který zajišťuje vstup od uživatele, pak je hodnotou tohoto parametru právě získaný vstup.

6.2.2.2 Třída CallFlow Tato třída slouží jako bázová třída pro implementaci objektu, který představuje kontejner pro jednotlivé stavy. Primárním úkolem této třídy je vytvořit a zaregistrovat jednotlivé stavy (viz 6.2.2.3) pomocí metody `addState`, resp. pomocí metody `addStates`, která registruje více stavů v jednom volání. Registrace stavů se provádí v metodě `create`, kterou musí každá instance implementovat.

[14] Technologie WebSockets byla představena v kapitole 5.2.6.

Tento kontejner zároveň slouží jako jakási sdílená sběrnice, která je přístupná všem stavům, které jsou v kontejneru obsaženy. Může tedy např. sloužit k uchování dat, která jsou společná pro více stavů.

Důležitou metodou této třídy je metoda `fireEvent(event, data)`. Tato metoda se volá v momentě, kdy je potřeba přejít do dalšího stavu. Volání přebírá dva parametry. Parametr `event` identifikuje typ události. Na základě typu události metoda rozhodne, do jakého dalšího stavu se má přejít. Každou vyvolanou událost můžou volitelně doprovázet `data` specifická pro danou událost.

Kód ve výpise 5 demonstruje vytvoření instance třídy `CallFlow` a registraci stavů. Takovou instanci je možné předat hlavnímu objektu aplikace prostřednictvím hodnoty parametru `controller` (viz 6.2.2.1).

6.2.2.3 Třída State Tato třída reprezentuje jeden stav aplikace. Jedná se o bázovou třídu, ze které se dají odvodit specifičtější třídy nebo lze přímo vytvořit instanci této třídy. Jednotlivé stavy se registrují do instance třídy `CallFlow` např pomocí metody `addState` (viz 6.2.2.2).

Konstruktor může přijímat tři parametry. Prvním a jediným povinným parametrem je parametr `id`. Jedná se o identifikátor stavu, který musí být jedinečný v rámci jednoho flow.

Zbývající parametry `target` a `event` jsou nepovinné. V případě, že jsou tyto parametry uvedeny, tak konstruktor třídy interně zavolá metodu `addTransition` s hodnotami těchto parametrů. Tato metoda má následující předpis `addTransition(event, target, condition)` a slouží k definici přechodů do dalších stavů.

Každý stav může mít definován více přechodů. Přechod je definován kombinací typu události a cílového stavu. Tato kombinace může být volitelně rozšířena ještě parametrem `condition`. Metodě `addTransition` může být cílový stav `target` předán formou reference na instanci třídy `State` nebo můžeme metodě předat pouze identifikátor stavu.

V některých případech se nám může hodit nadefinovat více přechodů se stejným typem události. Více přechodů se stejným typem události bývá typicky definováno u stavů, které nějakou formou implementují menu.

V momentě, kdy je proveden výběr v menu, může být vyvolána obecná událost, např. `continue`. Tato obecná událost je parametrizována hodnotou, která byla v menu zvolena. Zde se dostáváme k volitelnému parametru `condition`, který slouží k rozhodnutí, do jakého stavu se přejde. Hodnotou tohoto parametru je funkce, která přebírá jeden parametr a to je právě hodnota, která byla v menu zvolena. Tato funkce obsahuje rozhodovací logiku, která má tedy k dispozici informaci o vybrané hodnotě a vrací hodnotu `true` v případě, že se má daný přechod použít pro přechod do dalšího stavu. Demonstrace tohoto postupu je vidět ve výpise 5.

Třída dále obsahuje metody `addOnEntryAction` a `addOnExitAction`, které slouží k registraci funkcí, které se vyvolávají při vstupu, resp. při výstupu z tohoto stavu. Těchto funkcí může být zaregistrováno více. V případě, že je funkcí zaregistrováno více, jsou jednotlivé funkce volány sekvenčně za sebou. Každá taková funkce má v momentě zavolání k dispozici reference na následující objekty. Tyto reference jsou k dispozici

prostřednictvím argumentů funkce. Prvním argumentem je instance kontejneru (flow), do kterého daný stav patří. Druhým argumentem je reference na stav samotný. Poslední argument obsahuje doplňující data, která jsou závislá na konkrétním případě použití.

Jak již bylo předesláno dříve, každý stav může obsahovat vnořené stavy. K registraci vnořených stavů slouží metoda `addNestedCallFlow(callFlow)`, která přijímá jeden parametr. Hodnota parametru `callFlow` je instance, pro nás již známé, třídy `CallFlow`, která byla popsána v kapitole 6.2.2.2. Díky této možnosti můžeme jednoduše docílit znovupoužitelnosti kódu, kdy opakovaně využívanou komponentu implementujeme jako samostatnou třídu, která je odvozena ze třídy `CallFlow`. Instance této třídy je poté registrována zavoláním metody `addNestedCallFlow` na daném stavu. V případě potřeby může být tato instance parametrizována předáním parametrů do konstruktoru.

Třída `State` obsahuje tovární statickou metodu `create`, která je další možností, jak vytvořit instanci stavu. Oproti vytvoření stavu pomocí konstruktoru nabízí pohodlnější cestu k vytvoření stavu, který obsahuje hlasový model (viz 6.2.2.4). Pomocí hlasového modelu definujeme typ stavu. Jedním z typů je stav, který zajišťuje přečtení vstupu od uživatele, dalším typem stavu může být stav, který slouží k prezentaci výstupu. Vytváření stavů pomocí teto metody je použito ve výpise 5.

Ve stejném výpise si můžeme všimnout, že volání většiny metod třídy `State` se dá řetězit. V jednom řetězu volání můžeme vytvořit instanci stavu, navěsit akce, které se volají při přechodu do dalších stavů a nadefinovat přechody do dalších stavů.

6.2.2.4 Třída VoiceModel Jedná se o abstraktní třídu, ze které se instance nevytvářejí přímo. Slouží jako bázová třída, ze které jsou odvozeny třídy reprezentující konkrétní hlasový model.

Hlasový model představuje způsob, jakým definujeme typ stavu. Z tohoto pohledu knihovna rozlišuje dva typy stavů resp. hlasových modelů.

Prvním typem je hlasový model, který slouží k získání vstupu od uživatele. Takový hlasový model v praxi vypadá tak, že obsahuje text zprávy, pomocí které se uživateli sdělí, jaké jsou přípustné hodnoty pro vstup. Dále hlasový model v tomto případě obsahuje popis gramatiky, která definuje validní hodnoty pro vstup. Tento typ hlasového modelu je implementován třídou `Ask` (viz 6.2.2.9).

Dalším typem hlasového modelu, který také zajišťuje získání vstupu, je hlasový model představovaný třídou `Record` (viz 6.2.2.8). Tento hlasový model zaznamenává vstup formou nahrávky zprávy.

Druhým typem hlasového modelu je model, který slouží k prezentaci nějakého výstupu. V praxi takový hlasový model obsahuje pouze definici zprávy, která se má uživateli sdělit. Tento typ hlasového modelu je implementován třídou `Say` (viz 6.2.2.6) nebo třídou `Exit` (viz 6.2.2.7).

Instance tříd, které jsou z této třídy odvozeny obsahují všechny informace, které jsou nutné pro automatické vygenerování výsledného VXML souboru.

6.2.2.5 Třída Prompt Třída `Prompt` slouží k definici zprávy, která se má uživateli předložit. Tuto definici zprávy používají všechny třídy, které implementují hlasový model (viz kapitola 6.2.2.4).

Třída obsahuje kolekci `audios`. Tato kolekce může obsahovat jednu nebo více zpráv, které se uživateli postupně přehrají. Položkami této kolekce může být text, který bude pomocí TTS enginu převeden na hlasovou zprávu nebo reference na audio nahrávku, která bude přehrána.

Konstruktor třídy může být volán více způsoby. Ve výpise 7 uvádím ukázky volání konstruktoru.

```
1  var simplePrompt = new Prompt('This is prompt nr. 1.');
2
3  var combinedPrompt = new Prompt([
4    'This is prompt nr. 2.',
5    new Audio('/audio/message.wav')
6  ]);
7
8  var promptWithSettings = new Prompt({
9    text: 'This is prompt message.',
10   audios: [
11     'This is prompt nr. 2.',
12     new Audio('/audio/message.wav')
13   ],
14   bargein: false,
15   bargeinType: 'none',
16   timeout: 5,
17   language: 'en-US'
18 });
```

Výpis 7: Vytvoření instance třídy `Prompt`

Nejjednodušší možnost vytvoření instance je ukázáno na prvním řádku výpisu. Konstrukce spočítá v předání jediného argumentu, který definuje text zprávy, který bude pomocí TTS přehrán.

Na řádku 3 je ukázána možnost vytvoření instance předáním kolekce zpráv. Kolekce v tomto případě obsahuje text a referenci na audio nahrávku (více informací o třídě `Audio` najdete k kapitole 6.2.2.13).

Pokud potřebujeme vytvořit instanci a zároveň nastavit její další vlastnosti, tak můžeme využít volání konstruktoru, které je uvedeno na řádku 8. V tomto případě konstruktor přebírá jako jediný parametr objekt, který specifikuje veškerá nastavení instance.

Obě nastavení `text` a `audios` nastavují zprávu, která se uživateli předloží. V praxi nenastavujeme oba parametry, ale pouze jeden z nich. Nastavení `text` přebírá text zprávy a používá se tehdy, když definujeme pouze jednoduchou textovou zprávu. Hodnotou nastavení `audios` je kolekce zpráv, která může obsahovat textové zprávy nebo reference na audio nahrávky.

Hodnotou nastavení `bargein` definujeme, zda je možné zprávu během přehrávání přerušit stiskem klávesy nebo hlasovým vstupem. Výchozí hodnotou tohoto nastavení je `true`. Hodnota `true` povoluje přerušení přehrávání zprávy. Pokud chceme uživatele donutit, aby si musel zprávu vyslechnout celou, tak nastavíme hodnotu `false`.

Nastavením `bargeinType` blíže specifikujeme způsob, jakým je možné přehrávání zprávy přerušit. Hodnotou nastavení může být `speech` nebo `hotword`.

Nastavení `language` zprostředkovává TTS modulu informaci o jazyku, v jakém má být zpráva interpretována. Pokud není hodnota specifikována, přebírá se hodnota, kterou má nastavena IVR platforma.

Nastavením `timeout` můžeme specifikovat časovou prodlevu, po kterou bude VXML interpretr čekat na odezvu od uživatele. Hodnotou je číslo udávající tuto prodlevu ve vteřinách. Toto nastavení má smysl specifikovat v případě, kdy definujeme zprávu, která bude předcházet uživatelskému vstupu.

6.2.2.6 Třída Say Tato třída je odvozena ze třídy `VoiceModel` a implementuje tedy konkrétní hlasový model. Prostřednictvím tohoto hlasového modelu sdělujeme uživateli nějakou zprávu. Způsoby vytvoření instance této třídy jsou ukázány ve výpise 8.

```
1  var simpleSay = new Say('This is message.');
2
3  var sayWithSettings = new Say({
4    prompt: 'This is message.',
5    // nebo
6    prompt: new Prompt({
7      text: 'This is message.',
8      bargein: false
9    })
10 });
```

Výpis 8: Vytvoření instance třídy `Say`

Třída `Say` interně obsahuje instanci třídy `Prompt` a nastavuje hodnoty některým zděděným vlastnostem z třídy `VoiceModel`.

Ukázka vytvoření stavu, který obsahuje hlasový model tohoto typu, je vidět ve výpise zdrojového kódu ukázkové aplikace na řádku číslo 11.

Ve výpise 9 je ukázka VXML kódu, který pro takový stav aplikace automaticky vygeneruje.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vxml version="2.1">
3    <var name="event" expr="'continue'" />
4    <var name="sessionid" expr="'28'" />
5
6    <form id="greeting">
```

```
7       <block>
8         <prompt>Welcome to menu example.</prompt>
9         <submit next="/menu-example/stateMachine" namelist="
              ↪ event sessionid" />
10      </block>
11    </form>
12  </vxml>
```

Výpis 9: Kód vygenerovaný pro stav s hlasovým modelem `Say`

6.2.2.7 Třída Exit Třída `Exit` implementuje hlasový model, který uživateli přehraje zprávu a poté ukončí provádění aplikace a dojde tak k zavěšení hovoru.

Nejjednodušší ukázka vytvoření instance této třídy je uvedena ve zdrojovém kódu ukázkové aplikace na řádcích 14, 17, 20 a 23. Na těchto řádcích je uveden kód, který vytváří konečné stavy aplikace. Při vstupu do takového stavu dojde k prezentaci zprávy a následnému zavěšení hovoru.

Ve výpise 10 je uvedena ukázka VXML kódu, který je pro takový stav aplikací automaticky vygenerován.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vxml version="2.1">
3     <form id="optionOne">
4         <block>
5            <prompt>You selected option one.</prompt>
6            <exit />
7         </block>
8     </form>
9  </vxml>
```

Výpis 10: Kód vygenerovaný pro stav s hlasovým modelem `Exit`

6.2.2.8 Třída Record Tato třída implementuje typ hlasového modelu, který slouží k zaznamenání vstupu uživatele v podobě audio nahrávky. Uživateli je nejdříve přehrána zpráva, která ho vyzývá k zadání vstupu a poté dochází ke spuštění nahrávání. Konec nahrávání uživatel indikuje stiskem klávesy #.

Pořízení zvukové nahrávky má v kompetenci IVR platforma a musí tedy tuto funkcionalitu podporovat. Nutno podotknout, že tuto funkcionalitu nepodporuje IVR platforma VoiceGlue.

Jelikož práce s tímto hlasovým modelem není demonstrována ve zdrojovým kódu ukázkové aplikace, tak ve výpise 11 uvádím ukázku zdrojového kódu, který vytváří stav, jehož hlasovým modelem je instance této třídy. V ukázce jsou demonstrovány dva způsoby zavolání konstruktoru.

První volání spočívá v předání jednoho parametru. Tento parametr může být textový řetězec zprávy nebo instance typu `Prompt`. Z ukázky je dále vidět způsob, jakým je možné nahrávku uložit na disk.

Druhé volání také přebírá jeden parametr. Hodnotou je objekt, který obsahuje jednotlivá nastavení.

Nastavení `prompt` je zpráva, která bude uživateli předložena.

Nastavení `maxTime` nastavuje maximální délku nahrávky ve vteřinách. Výchozí hodnotou tohoto nastavení je 60 (s), tedy jedna minuta.

Nastavení `beep` udává, zda má po přehrání zprávy dojít k přehrání pípnutí, které indikuje začátek procesu nahrávání. Ve výchozím stavu je toto nastavení zapnuto.

Dalším nastavením je `finalSilence`. Tímto nastavením můžeme nadefinovat dobu v milisekundách, po kterou musí uživatel vydržet tiše, aby tuto pauzu VXML interpretr vyhodnotil jako konec zadání uživatelova vstupu a ukončil tak proces nahrávání. Výchozí hodnotou tohoto nastavení je 2500 (ms).

Posledním možným nastavením je `type`. Hodnotou tohoto nastavení udáváme MIME typ nahraného souboru. Pod tímto MIME typem bude audio nahrávka odeslána serveru[15]. Výchozí hodnotou je `audio/wav`.

```
1  var recording =
2    new vxml.Record('Record your message after the beep.');
3
4  // NEBO
5
6  var recording =
7    new vxml.Record({
8      prompt: 'Record your message after the beep.',
9      maxTime: 60,
10     beep: true,
11     finalSilence: '2500ms',
12     type: 'audio/wav'
13   });
14
15 var recordingState = vxml.State.create('getRecording',
     ↪ recording)
16   .addOnExitAction(function* (cf, state, event) {
17     var recordingDataBuffer = event.data;
18     var path = __dirname + '/recordings/recording.wav';
19
20     yield writeFile(recordingDataBuffer, path);
21   });
```

Výpis 11: Vytvoření stavu s hlasovým modelem `Record`

[15]Konkrétně je hodnota MIME typu přítomna v hlavičce Content-Type požadavku.

Ve výpise 12 je pro představu uveden VXML kód, který aplikace pro takový stav automaticky vygeneruje.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <vxml version="2.1">
3    <var name="event" expr="'continue'"></var>
4    <var name="sessionid" expr="'28'"></var>
5
6    <form id="getRecording">
7      <record name="callersMessage" beep="true" maxtime="60s"
8        finalsilence="2500ms" type="audio/wav">
9        <prompt>Record your message after the beep.</prompt>
10     </record>
11
12     <filled>
13       <submit expr="'/app/saveRecording?event=' + event +
             ↪ '&amp;sessionid=' + sessionid" method="post"
             ↪ namelist="callersMessage" enctype="multipart/
             ↪ form-data"/>
14     </filled>
15   </form>
16 </vxml>
```

Výpis 12: Kód vygenerovaný pro stav s hlasovým modelem `Record`

6.2.2.9 Třída Ask Třída `Ask` představuje hlasový model, který slouží k zajištění vstupu od uživatele. Částečně je tato třída podobná třídě `Say`. S touto třídou má společné to, že uživateli předkládá nějakou zprávu - interně tedy také pracuje s instancí třídy `Prompt`.

Jelikož tento hlasový model zajišťuje uživatelský vstup, je potřeba mít k dispozici mechanismus, pomocí kterého bychom byli schopni popsat, jak mají hodnoty na vstupu vypadat. Třída interně využívá pro tento účel gramatiky, které byly představeny v kapitole 2.4.

Pro účely definice těchto gramatik je využito několika tříd, kterým se budu podrobněji věnovat v dalších částech textu. Ve stručnosti zde jen uvedu, že máme k dispozici třídu `BuiltinGrammar` pro definici vestavěných gramatik, třídu `Grammar` pro definici DTMF a hlasových gramatik, a třídu `Choices`, která slouží k definici smíšených hlasových a DTMF gramatik.

Ve výpise 5 se zdrojovým kódem ukázkové aplikace je na řádku 26 uveden kód, který vytváří stav, jehož hlasovým modelem je právě instance třídy `Ask`. Z kódu je patrné, že konstruktor třídy přijímá jeden objekt. Tento objekt musí definovat minimálně dvě povinná nastavení.

Jedno z těchto nastavení je `prompt`, které slouží k definici předkládané zprávy. Tato zpráva by měla uživatele informovat o tom, jaké jsou přípustné hodnoty pro vstup. Typem tohoto nastavení je buď řetězec nebo instance třídy `Prompt`.

Dalším povinným nastavením, které musíme při konstrukci objektu uvést, je nastavení `grammar`. Toto nastavení přebírá instanci třídy, která definuje přípustné hodnoty pro vstup.

Při vytváření instance můžeme volitelně nastavit zprávy, které se uživateli předloží v situacích, kdy nedošlo k zadání očekávaného vstupu.

Pomocí nastavení `noInputPrompts` můžeme nastavit znění zprávy, která bude uživateli přehrána v situaci, kdy nedošlo k zadání žádného vstupu.

Nastavením `noMatchPrompts` definujeme zprávu, která se uživateli přehraje v momentě, kdy byl nějaký vstup zadán, ale nejedná se o validní vstup popsaný gramatikou.

Obě tyto nastavení jsou typu pole a můžou tedy obsahovat více zpráv pro každý typ nastavení.

Pokud je definována jen jedna zpráva, dojde k přehrání této zprávy a následně dojde k vyvolání události `noinput`, resp. `nomatch`. Na tyto události může aplikace dále reagovat např. přechodem do chybného stavu.

Pokud je hodnotou nastavení pole více zpráv, tak zpracování funguje následovně. Po prvním neúspěšném zadání vstupu dojde k přehrání první zprávy. Poté po každém dalším špatném zadání dojde k přehrání další zprávy v pořadí. Tohoto se dá využít například k postupnému přehrávání podrobnějších informací.

Pro stav, který má jako hlasový model uvedenu instanci třídy `Ask`, aplikace automaticky vygeneruje VXML kód, který je předmětem výpisu 13.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <vxml version="2.1">
3    <var name="event" expr="'continue'" />
4    <var name="result" expr="" />
5    <var name="sessionid" expr="'28'" />
6
7    <form id="menu">
8      <field name="F_1" type="digits?length=1">
9        <prompt>Press one, two or three.</prompt>
10     </field>
11
12     <filled>
13       <assign name="result" expr="F_1.out || F_1" />
14       <submit next="/menu-example/stateMachine" namelist="
           ↪ event result sessionid" />
15     </filled>
16
17     <noinput count="1">
18       <assign name="event" expr="'noinput'" />
```

```
19          <submit next="/menu-example/stateMachine" namelist="
               ↪ event result sessionid" />
20        </noinput>
21
22        <nomatch count="1">
23          <assign name="event" expr="'nomatch'" />
24          <submit next="/menu-example/stateMachine" namelist="
               ↪ event result sessionid" />
25        </nomatch>
26      </form>
27    </vxml>
```

Výpis 13: Kód vygenerovaný pro stav s hlasovým modelem `Ask`

6.2.2.10 Třída Grammar Prostřednictvím instancí této třídy definujeme hlasové nebo DTMF gramatiky. V podstatě se jedná o abstrakci nad VXML elementem `<grammar>`, který byl představen v kapitole 2.4.

Tato třída je úzce spjata s třídou `Ask`, která tuto třídu využívá k definici přípustných vstupních dat. Pomocí této třídy lze specifikovat pouze gramatiky, které definují uživatelský vstup jako výběr jedné položky z předem definované konečné množiny.

Třída zároveň podporuje externí gramatiky, tedy gramatiky, jejichž definice se nachází v samostatném XML souboru.

Ve výpise 14 je vidět zdrojový kód, vytvářející stav, jehož úkolem je načíst vstup od uživatele. Povolené hodnoty vstupu jsou definovány pomocí gramatiky. V ukázce je dále vidět způsob, jakým lze programově přistoupit k zadanému vstupu.

```
1  var yesNoGrammar = new Grammar({
2    mode: 'voice',
3    items: [
4      { text: 'yes', tag: 'yes' },
5      { text: 'yeah', tag: 'yes' },
6      { text: 'ya', tag: 'yes' },
7      { text: 'no', tag: 'no' },
8      { text: 'nope', tag: 'no' }
9    ]
10 });
11
12 var getInput = new Ask({
13   prompt: 'Are you happy?',
14   grammar: yesNoGrammar
15 });
16
17 var getInputState = vxml.State.create('getInput', getInput)
```

```
18    .addOnExitAction(function* (cf, state, event) {
19      var answer = event.data;
20
21      if (answer == 'yes') {
22        console.log('You are happy.');
23      }
24      else {
25        console.log('You are not happy.');
26      }
27    });
```

Výpis 14: Vytvoření hlasové gramatiky

Na řádcích 1 až 10 dochází k vytvoření instance gramatiky. Můžeme si všimnout, že konstruktor přebírá jeden parametr. Hodnotou tohoto parametru je objekt obsahující veškerá potřebná nastavení.

Prvním uvedeným nastavením je `mode`. Možné hodnoty nastavení jsou "voice" nebo "dtmf". Tento údaj specifikuje, zda vytváříme hlasovou nebo DTMF gramatiku.

Další nastavení je `items`. Jedná se o pole položek, kde každá položka obsahuje objekt s vlastností `text` a `tag`. Vlastnost `text` definuje text, který se bude ASR modul snažit rozpoznat. Pokud dojde k úspěšnému rozeznání tohoto textu, je z gramatiky vrácena návratová hodnota. Návratová hodnota je obsažena ve vlastnosti `tag`. S návratovou hodnotou lze dále pracovat. Lze ji například uložit do databáze nebo na základě této hodnoty rozvětvit program. Více položkám můžeme přiřadit stejný tag.

Definice DTMF gramatiky by vypadala analogicky. Při vytváření instance bychom pouze nastavili hodnotu nastavení `mode` na "dtmf". Následně bychom v definici jednotlivých položek pole `items` mohli jako hodnoty vlastnosti `text` použít pouze následující hodnoty: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #.

6.2.2.11 Třída BuiltinGrammar

Tato třída představuje vestavěné gramatiky. Vestavěné gramatiky se od klasických gramatik liší způsobem, jakým jsou zapsány ve výsledném VXML kódu. Standard VXML definuje sadu vestavěných gramatik, které není v kódu potřeba nijak dále specifikovat. Stačí uvést typ gramatiky a případně její parametry. Vestavěné gramatiky byly představeny v kapitole 2.4.2.

Ve výpise 5 je na řádcích 28-31 ukázán způsob, jakým lze vytvořit instanci této třídy.

6.2.2.12 Třída Choices

Tuto třídu je užitečné použít v případech, kdy chceme uživateli nabídnout možnost zadání vstupu libovolnou metodou. Uživatel může zadat vstup pomocí hlasové volby nebo pomocí DTMF volby. Na základě popisu třída interně vytvoří dvě instance třídy `Grammar`. První instance představuje hlasovou gramatiku a druhá instance představuje DTMF gramatiku.

Ve výpise 15 se nachází zdrojový kód, který vytváří instanci kombinované gramatiky `Choices`.

```
1  var yesNoChoices = new vxml.Choices([
2    {
3      items: ['yes', 'ya', 'dtmf-1', 'dtmf-*'],
4      tag: 'yes'
5    },
6    {
7      items: ['no', 'nope', 'dtmf-2', 'dtmf-#'],
8      tag: 'no'
9    }
10 ]);
11
12 var getInput = new Ask({
13   prompt: 'Are you happy?',
14   grammar: yesNoChoices
15 });
16
17 // ... vytvoreni stavu a precteni vstupnich dat
```

Výpis 15: Vytvoření kombinované gramatiky `Choices`

Zápis definice je hodně podobný způsobu zápisu, který se používá u definice gramatik prostřednictvím třídy `Grammar`. Konstruktor třídy `Choices` přebírá jeden parametr a to je pole položek, kde každá položka obsahuje dvě vlastnosti. První vlastnost je `items` a jedná se o pole řetězců. Jednotlivé řetězce specifikují texty, které bude rozpoznávat ASR modul nebo kódy DMTF volby. Pokud chceme definovat položku jako DMTF volbu, tak ji musíme prefixovat řetězcem "dtmf-", např. tedy "dtmf-1". Vlastnost `tag` specifikuje návratovou hodnotu z gramatiky.

6.2.2.13 Třída Audio Prostřednictvím této třídy lze vytvářet reference na audio soubory, které se dají použít jako součást zprávy uvnitř kontejneru `Prompt`. Jedná se o abstrakci nad VXML elementem `<audio>`.

Vytvoření instance třídy je přímočaré. Ve výpise 16 je uveden kód, který vytváří instanci jako součást definice zprávy.

```
1 var promptWithAudio = new Prompt([
2   new Audio('/audio/welcome_message.wav', 'Welcome')
3 ]);
```

Výpis 16: Vytvoření instance třídy `Audio`

Konstruktor této třídy přijímá dva parametry. Hodnotou prvního parametru je cesta k audio souboru, který se má přehrát. Hodnota druhého parametru specifikuje text, který bude přehrán TTS modulem v případě, že se nepodaří specifikovaný audio soubor stáhnout. Tento parametr není povinný.

Textová hodnota	Délka pauzy [s]
none	0
x-weak	0,35
weak	0,35
medium	0,7
strong	1
x-strong	1

Table 3: Textové hodnoty pro definici pauzy a přidružených délek pauzy

6.2.2.14 Třída Silence Tato třída se využívá k vytvoření pauzy mezi přehrávanými částmi zprávy. Jedná se o abstrakci nad VXML elementem `<break>`.

Obsahem výpisu 17 je ukázka zdrojového kódu, který demonstruje vytvoření krátké pauzy mezi přehrávanými částmi zprávy. Z ukázky je patrné, že je možné délku pauzy specifikovat více způsoby. Jedním způsobem je použít jednu z definovaných textových hodnot. V tabulce 3 uvádím seznam hodnot, které lze použít. Současně s těmito hodnotami je v tabulce pro každou hodnotu uvedena přidružená délka pauzy[24].

Pokud nám pro definici délky pauzy nepostačují textové hodnoty, můžeme pauzu vyjádřit ve vteřinách nebo v milisekundách.

Oba způsoby definice pauzy jsou znázorněny ve výpise zdrojového kódu 17.

```
1 var promptWithSilence = new Prompt([
2   'Hello',
3   new Silence('strong'),
4   'my name is',
5   new Silence('200ms'),
6   'Ondrej',
7   new Silence('1s')
8 ]);
```

Výpis 17: Vytvoření instance třídy `Silence`

6.2.2.15 Třída SayAs Tato třída je abstrakcí nad VXML elementem `<say-as>`. Umožňuje nadefinovat způsob, jakým bude text interpretován TTS modulem. Tímto způsobem můžeme TTS modul instruovat například o tom, aby číslo 1234 neinterpretoval jako souvislé číslo, ale aby výsledkem byla postupná interpretace jednotlivých číslic, tedy 1, 2, 3, 4. Ve výpise 18 je demonstrováno použití třídy `SayAs`.

```
1 var promptWithSayAs = new Prompt([
2   'Your PIN is ',
3   new SayAs('1234', 'digits')
4 ]);
```

Výpis 18: Vytvoření instance třídy `SayAs`

Konstruktor třídy přebírá dva parametry. Hodnotou prvního parametru je text, který se má interpretovat, druhý parametr udává typ interpretace. Důležité je upozornit na to, že je tato funkcionalita závislá na použitém TTS modulu a některé typy interpretací nemusí být dostupné. V následujícím seznamu uvádím typy interpretací, které by měly být podporovány TTS modulem, který je součástí IVR platformy Voxeo Prophecy[24]:

- time
- boolean
- date
- digits
- currency
- number
- phone

6.2.2.16 Třída Var Instance této třídy slouží jako zástupné objekty pro hodnoty, které nejsou známy v době registrace stavů do kontejneru. Typicky se jedná o primitivní hodnoty, které se v JavaScriptu předávají hodnotou. Na primitivní hodnotu není možné získat referenci, a proto je potřeba takovou hodnotu zabalit do zástupného objektu.

Ve výpise 19 je demonstrována situace, kdy je použití zástupného objektu nevyhnutelné.

```
1  var GetAgeCallFlow = CallFlow.extend({
2    constructor: function () {
3      // zavolani konstruktoru bazove tridy
4      GetAgeCallFlow.super.call(this);
5
6      // tato hodnota neni dostupna dokud nedojde k pruchodu
       ↪ prvnim stavem
7      this._age = null;
8    },
9
10   create: function* () {
11     var getInputModel = new Ask({
12       prompt: 'Enter your age.',
13       grammar: new BuiltinGrammar({
14         type: 'digits', minLength: 1
15       })
16     });
17
18     var confirmInputModel = new Say([
```

```
19          'You are ',
20          new Var(this, '_age'),
21          ' years old.'
22        ]);
23
24        var confirmInputState =
25          State.create('confirmInput', confirmInputModel);
26
27        var getInputState = State.create('getInput',
            ↪ getInputModel)
28          .addTransition('continue', confirmInputState);
29          .addOnExitAction(function* (cf, state, event) {
30            // ulozeni vstupu do vlastnosti _age
31            cf._age = event.data;
32          });
33
34        this
35          .addState(getInputState)
36          .addState(confirmInputState);
37    }
38  });
```

Výpis 19: Použití zástupného objektu `Var`

Výše uvedený kód vytváří flow aplikace, které se skládá ze dvou stavů. První stav slouží k zjištění věku uživatele. Druhý stav slouží k zobrazení sdělení o zadaném věku. A proto potřebuje hlasový model druhého stavu referencovat hodnotou vlastnosti `this._age`. Tato hodnota ale není ve fázi vytváření flow dostupná. Hodnota bude dostupná až v momentě, kdy dojde k průchodu prvním stavem. Z tohoto důvodu chybějící hodnotu zastupuje zástupný objekt reprezentovaný instancí třídy `Var` (viz řádek 20).

Konstruktor této třídy přijímá dva parametry. Hodnotou prvního parametru je kontext. Kontext představuje objekt, ze kterého se bude číst vlastnost specifikovaná druhým parametrem konstruktoru. V tomto případě je typem druhého parametru řetězec.

V některých případech může být vhodnější předat funkci jako hodnotu druhého parametru. Tato funkce může obsahovat například nějakou formátovací logiku (viz výpis 20).

```
1  // uvnitr metody create
2  // predpokladame existanci instancni vlastnosti _age
3  var confirmInputModel = new Say([
4    'You are ',
5    new Var(this, function () {
6      if (this._age > 25) {
7        return 'so many';
```

```
8       }
9       else {
10        return this._age;
11      }
12    }),
13    ' years old.'
14  ]);
```

Výpis 20: Použití zástupného objektu `Var` s funkčním parametrem

Třída definuje metodu `getValue`. Tato metoda je zavolána v momentě, kdy je potřeba vyhodnotit skutečnou hodnotu zástupného objektu.

Pokud byla jako druhý parametr volání konstruktoru předána funkce, tak dojde k zavolání této funkce v kontextu[16], který byl specifikován prvním parametrem. Návratová hodnota této funkce potom představuje hodnotu zástupného objektu.

Pokud byl hodnotou druhého argumentu řetězec, pak dojde k nalezení pojmenované vlastnosti v rámci specifikovaného kontextu.

6.2.3 Popis implementace vybraných částí hlasového portálu

V předchozí kapitole jsme získali detailní informace o struktuře a fungování navržené knihovny.

V této kapitole se opět vrátím k hlasovému portálu rezervačního systému a zaměřím se zde na popis vybraných komponent, které jsou nějakým způsobem zajímavé a stojí za to zmínit se o nich podrobněji.

Všechny tyto popisované komponenty jsou navrženy s ohledem na znovupoužitelnost kódu. Principu znovupoužitelnosti se dosahuje zaregistrováním instance komponenty jako sady vnořených stavů (viz kapitola 6.2.2.3).

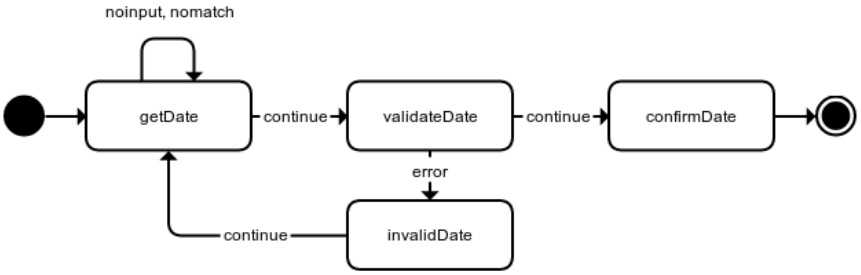
6.2.3.1 Komponenta pro zadání data Tato komponenta slouží k zadání data pomocí DTMF volby. V aplikaci hlasového portálu se tato komponenta používá v menu pro filtrování seznamu letů. Prostřednictvím této komponenty je uživatel vyzván k zadání požadovaného data odletu, resp. příletu.

Obrázek 10 znázorňuje strukturu této komponenty formou stavového diagramu. V aplikaci je tato komponenta reprezentována třídou `GetDateDtmfFlow`.

První stav $getDate$ uživateli předkládá zprávu o tom, že se od něj očekává zadání data formou osmi číslic ve formátu $ddmmyyyy$. Prostřednictvím hodnoty parametru předané do konstruktoru je možné pro každou instanci této komponenty nastavit různý text této zprávy. Komponenta zůstává v tomto stavu, dokud nedojde k zadání vstupu v požadovaném formátu.

Pokud došlo k zadání vstupu v požadovaném formátu (8 číslic), tak zpracování přechází do druhého stavu $validateDate$. Tento stav má za úkol ověřit, zda je zadaný

[16] V JavaScriptu může být funkce zavolána v kontextu, který může být programově nastaven na specifický objekt. Reference `this` uvnitř funkce poté ukazuje na tento nastavený objekt (kontext).



Obrázek 10: Struktura komponenty pro zadání data

vstup validním datem. Ověřuje se např. zda zadaný den nepřesahuje za poslední den v daném měsíci. Toto ověření bere v úvahu i přestupný rok.

Pokud zadaný vstup neprošel všemi validačními pravidly, dochází k vyvolání události $error$ a zpracování tak přechází do stavu $invalidDate$. Hlasový model tohoto stavu uživatele informuje o tom, že zadal neplatný vstup. Po přehrání této informace uživatel automaticky přechází do prvního stavu $getDate$, kde je vyzván k opakovanému zadání.

V případě, že došlo k zadání validního data, provede se přechod ze stavu $validateDate$ do stavu $confirmDate$. Tento stav uživateli zrekapituluje zadané datum a přechází do konečného stavu, kde životní cyklus komponenty končí.

Komponenta implementuje veřejnou metodu `getDate`. Tato metoda vrací prostřednictvím návratové hodnoty zadané datum. Metoda je typicky volána v rámci akce `onExitAction` definované na stavu, jehož je komponenta součástí.

6.2.3.2 Komponenta pro zadání textového vstupu Na obrázku 11 je znázorněna struktura komponenty sloužící pro získání textového vstupu pomocí DTMF volby. Vstup se zadává stejným způsobem jako se na mobilním telefonu s klávesnicí píše SMS zpráva.

Struktura komponenty je velmi jednoduchá. Skládá se pouze z jednoho stavu $askForInput$. Tento stav uživateli přehraje zprávu, prostřednictvím které uživatele vyzve k zadání textu. Text této zprávy je konfigurovatelný parametrem, který se předává konstruktoru třídy. Stav poté očekává zadání vstupu. Konec zadávání vstupu se komponentě indikuje krátkou neaktivitou nebo stisknutím klávesy #.

Zpracování vstupu probíhá ve funkci, která je zaregistrovaná pro akci `onExitAction` (viz 6.2.2.3). Implementace této funkce spočívá v převedení vstupní číselné sekvence na textový formát. Převod se řídí mapováním, které je znázorněno v tabulce 4. Např. následující sekvence 2446665 je převedena na řetězec ”ahoj”.

V tabulce si můžeme všimnout speciálního významu klávesy 0 ($skipchar$). Tato klávesa slouží k ukončení zadávání znaku a přesunu na zadání dalšího znaku.

Stisk této klávesy není nutný v případech, kdy zadáváme dva za sebou jdoucí znaky, které jsou namapovány na stisky různých kláves.

Klávesa č.	1. stisk	2. stisk	3. stisk	4. stisk	5. stisk
0	*skipchar*	-	-	-	-
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

Table 4: Mapování stisků kláves na znaky

Obrázek 11: Struktura komponenty pro zadání textového vstupu

Stisk klávesy 0 je nutný, pokud potřebujeme zadat dva po sobě jdoucí znaky, kdy každý z nich je namapován na stejnou klávesu. Pokud bychom např. potřebovali zadat textový řetězec "abc", museli bychom ho zadat pomocí sekvence 20220222.

Třída `GetTextInputDtmfFlow` implementující tuto komponentu vystavuje veřejnou metodu `getInput`. Metoda vrací zadaný vstup převedený do textové podoby.

Tato komponenta typicky bývá součástí jiné komplexnější komponenty. Jedna taková komponenta je popsána v následujícím textu.

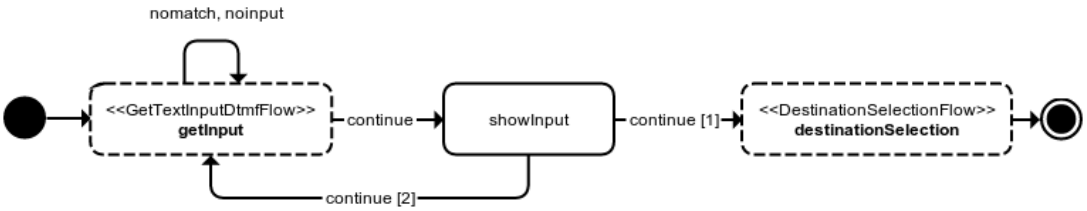
6.2.3.3 Komponenta pro zvolení destinace letu Struktura této komponenty je znázorněna v diagrumu 12. Komponenta je implementována třídou `GetDestinationFlow`.

Úkolem komponenty je vyhledání destinace a následné zvolení výběru z vyhledaného seznamu. Tato komponenta je součástí filtru letů a slouží k zadání odletové, resp. příletové destinace.

Stav $getInput$ uživatele vyzve k zadání řetězce na jehož základě se provede vyhledání destinací. Tento stav obsahuje komponentu, která tento vstup zajišťuje (komponenta byla popsána v kapitole 6.2.3.2).

V momentě, kdy máme k dispozici zadaný vstup, přechází zpracování do stavu $showInput$. V tomto stavu je uživateli přehrán text, který byl zadán. Následuje dotaz, zda zadaný vstup odpovídá tomu, co uživatel zamýšlel. Následně uživatel může zadaný vstup potvrdit stiskem klávesy 1 nebo odmítnout stiskem klávesy 2.

Pokud uživatel zadaný vstup odmítne, vrací se provádění do stavu $getInput$, kde má uživatel možnost provést nové zadání.



Obrázek 12: Struktura komponenty pro zvolení destinace

V momentě, kdy uživatel zadaný vstup potvrdí stiskem klávesy 1, přechází provádění do stavu $destinationSelection$. Tento stav obsahuje komponentu `DestinationSelectionFlow`, která tvoří vnitřní stavy tohoto stavu.

Komponenta představuje seznam vyhledaných destinací na základě vyhledávacího řetězce. Seznam vyhledaných destinací dostává komponenta k dispozici prostřednictvím reference, která je předána hodnotou parametru do konstruktoru. Struktura této komponenty je zobrazena na obrázku 13. Z diagramu je patrné, že struktura komponenty je závislá na počtu vyhledaných destinací a může mít dvě podoby.

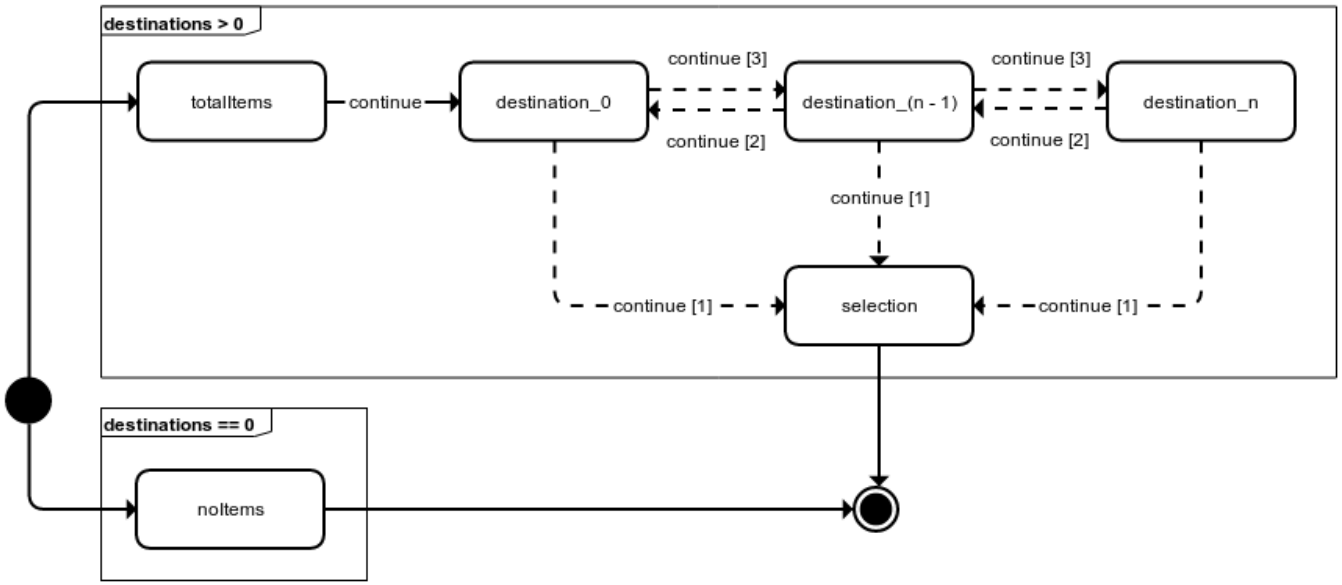
Pokud na základě vyhledávacího řetězce nedošlo k nalezení žádné destinace, tak komponenta obsahuje pouze jeden stav $noItems$. Tento stav, prostřednictvím svého hlasového modelu, uživateli sdělí, že žádné destinace nebyly nalezeny. Následně se provede přechod do konečného stavu, kde zpracování končí.

Pokud bylo nalezeno n destinací a $n > 0$, dojde k vytvoření n stavů. Pro každou z nalezených destinací je vytvořen jeden stav. Každý tento stav uživateli přehraje název destinace. Všechny tyto stavy jsou propojeny do řetězu, kterým je možné se pohybovat vpřed a vzad, tedy přecházet do předchozího a následujícího stavu. Tomuto řetězu předchází stav $totalItems$, který uživatele informuje o celkovém počtu nalezených destinací a o možnostech navigace mezi těmito záznamy.

Navigace je založena na DTMF volbě. K přechodu na předchozí záznam slouží klávesa 2. Pokud potřebujeme přejít k následujícímu záznamu, můžeme tak učinit stisknutím klávesy 3. Výběr aktuálního záznamu potvrzujeme stiskem klávesy 1. Stiskem této klávesy se dostáváme do stavu $selection$.

Stav $selection$ uživatele informuje o zvolené destinaci. Následuje přechod do konečného stavu, kde zpracování končí.

Komponenta implementuje metodu `getSelectedDestination`, která slouží k získání zvolené destinace. Tato metoda je typicky volána v rámci akce `onExitAction` definované na stavu, jehož je komponenta součástí.



Obrázek 13: Struktura stavu $destinationSelection$ ($DestinationSelectionFlow$)

7 Konfigurace a zprovoznění hlasového portálu

V této kapitole popíši kroky vedoucí k úspěšné konfiguraci a zprovoznění hlasového portálu na IVR platformách Voxeo Prophecy a VoiceGlue.

7.1 Aplikační server

Abychom mohli přistoupit ke konfiguraci jednotlivých IVR platforem, musíme nejdříve nainstalovat a spustit aplikační server. Spuštěním aplikačního serveru dostaneme přístup k webovému i hlasovému portálu rezervačního systému. Následuje popis instalace a spuštění aplikačního serveru v prostředí operačního systému Ubuntu.

Jak již bylo uvedeno, aplikační server je poháněn technologií Node.js (viz 5.2.3). Je tedy nutné, aby na stroji byl Node.js nainstalován. Popis instalace Node.js zde popisovat nebudu, jen uvedu odkaz na oficiální web[26], kde je možné nalézt instalační balíky. Pří výběru instalačního balíku je nutné zvolit minimálně verzi v0.11.2. Od této verze je součástí Node.js implementace generátorů[17], které jsou nutné pro spuštění serveru.

Další prerekvizitou je instalace MongoDB databáze. Popis instalace pro Ubuntu je detailně popsán na této URL[25].

V momentě, kdy máme výše uvedené prerekvizity nainstalovány, můžeme přejít k instalaci modulů, na kterých je běh aplikačního serveru závislý. Otevřeme si terminál a přepneme se do adresáře, kde máme rozbalený archiv se zdrojovými kódy aplikačního serveru. Nyní můžeme spustit následující příkazy:

- `npm install`
- `bower install`
- `npm install -g grunt-cli`

První z výše uvedených příkazů instaluje moduly, na kterých je závislá serverová část aplikace (např. Express.js). Druhý příkaz se postará o instalaci knihoven, které jsou vyžadovány klientskou částí aplikace (např. Angular.js). Poslední příkaz instaluje CLI nástroje Grunt. Tento příkaz není nutné spouštět v případě, že již máte Grunt na stroji nainstalován.

Nyní bychom měli mít nainstalováno vše, co je pro spuštění aplikačního serveru nutné. Ještě než přistoupíme k samotnému spuštění, bylo by vhodné v databázi vytvořit nějaké testovací záznamy. Spusťme proto následující příkazy:

- `grunt insertUsers`
- `grunt insertFlights`

[17] Generátory jsou použity v kódu VXML knihovny i aplikačního serveru pro přehlednější zpracování asynchronních operací.

Tyto příkazy v databázi vytvoří záznamy o dvou uživatelích, dále vytvoří deset záznamů o letech.

Nyní by nám již nemělo nic bránit ve spuštění aplikačního serveru. Spuštění provedeme příkazem `grunt`. Pokud spuštění proběhlo v pořádku, tak bychom měli v terminálu vidět vypsanou informaci o tom, že aplikace byla spuštěna a naslouchá na portu 9000.

Webový portál je k dispozici na URL `http://localhost:9000/flights`. Po zadání této URL do prohlížeče by se mělo zobrazit okno pro přihlášení. Pro vyzkoušení přihlášení můžeme zkusit zadat jako přihlašovací jméno a heslo řetězec "12345".

Hlasový portál je k dispozici na URL `http://localhost:9000/vxml`. Po zadání této URL do prohlížeče by se měla zobrazit chyba aplikace.

7.2 Voxeo Prophecy

Instalace a zprovoznění této IVR platformy je jednoduché. Vlastnosti této platformy byly blíže představeny v kapitole 2.2.1.

V následující části nejdříve popíši postup instalace a poté popíši jak provést konfiguraci hlasového portálu.

7.2.1 Instalace

Hned na začátku je důležité poznamenat, že z linuxových distribucí, Voxeo Prophecy podporuje jen ty založené na RedHatu, např. tedy CentOS. Instalační balík stáhneme zadáním následujícího příkazu do terminálu:

```
wget "http://voxeo.com/wp-content/themes/voxeo/inc/prophecy13
_download_linux64_tts_small.php"
```

Jakmile je soubor stažen, tak si zjistíme název staženého souboru a následujícím příkazem spustíme instalaci:

```
sh prophecy-13.0.5.20.74229-x64.bin -i console
```

Během instalace budeme vyzváni k potvrzení několika informací. Pokud jsme na konci instalace povolili automatický start služeb, tak bychom měli mít po dokončení instalace spuštěny tři nové služby:

- vdirectory
- vprism
- vserver

Zda jsou služby spuštěny můžeme ověřit spuštěním příkazu `./prophecy status`. Tento příkaz je nutné spouštět z umístění, ve kterém máme Voxeo Prophecy nainstalované. Pokud jsme při instalaci nic neměnili, tak se jedná o umístění `/opt/voxeo/prophecy`. Pokud služby spuštěny nejsou, můžeme je spustit příkazem `./prophecy start`.

7.2.2 Konfigurace

Nyní můžeme přistoupit ke konfiguraci hlasového portálu. Součástí IVR Voxeo Prophecy je webové konfigurační rozhraní, pomocí kterého konfiguraci provedeme.

Do tohoto rozhraní se dostaneme zadáním následující URL do internetového prohlížeče `http://localhost:9996/ManagementConsole`. Nyní bychom měli mít zobrazenu stránku s přihlašovacím formulářem. Přihlásíme se zadání řetězce "admin", které slouží jako přihlašovací jméno i heslo. Na obrázku 14 je zobrazena stránka, na které provedeme vytvoření nové VXML aplikace. Po přihlášení je nutné vybrat v horním menu vyznačenou ikonku (1.). Tímto se dostaneme na stránku, kde provedeme vytvoření nové aplikace. Nyní klikneme na ikonku + (2.). Tímto se nám ve spodní části stránky otevře formulář, který je nutné vyplnit.

Do pole $Name$ můžeme vyplnit jakýkoliv text. Jedná se pouze o název aplikace. Pole $Type$ specifikuje typ vytvářené aplikace, vybereme hodnotu "VXML". Do pole $URL1$ vyplníme URL adresu, na které je spuštěn hlasový portál (viz 7.1). Ve výchozím stavu je hlasový portál spuštěn na URL `http://localhost:9000/vxml`.

Posledním krokem je vytvoření adresy aplikace. Tuto adresu je nutné vyplnit, aby bylo možné se na aplikaci dovolat. Adresu vytvoříme přidáním nového záznamu do seznamu ve spodní části formuláře. Můžeme vyplnit např. řetězec "dp". Zbývá vyplněné nastavení uložit. Toto provedeme stisknutím tlačítka $Save$ (3.).

Nyní by měla být naše aplikace vytvořena. Můžeme tedy provést zkušební hovor. Součástí Voxeo Prophecy je testovací SIP klient. Tento klient běží v prostředí prohlížeče a dostaneme se k němu zadáním následující URL do prohlížeče `http://localhost:9994/vphone/VPhoneClient/main.html`. Po zadání této URL bychom měli vidět stránku, kde máme možnost vyplnit SIP adresu naší aplikace. Zadáme tedy `sip:dp@localhost` a stiskneme tlačítko $Dial$. Pokud je vše správně nastavené, tak bychom měli za chvíli slyšet uvítací zprávu hlasového portálu.

7.3 VoiceGlue

V této kapitole popíši instalaci a konfiguraci opensource IVR platformy VoiceGlue. Vlastnosti této platformy byly blíže představeny v kapitole 2.2.2.

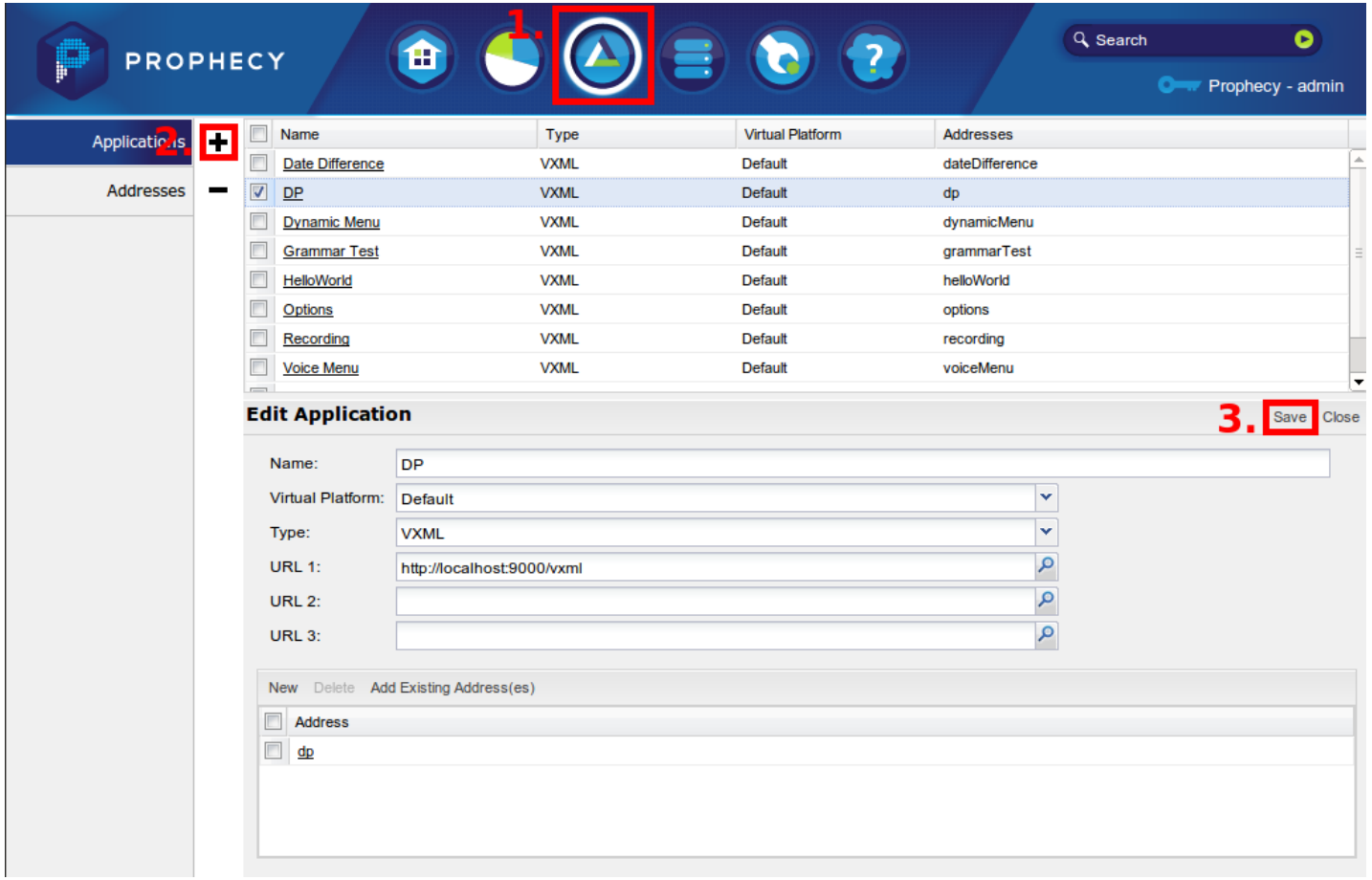
7.3.1 Instalace

Instalaci budeme provádět na stroji s operačním systémem Ubuntu Server. Něž přistoupíme k samotné instalaci IVR VoiceGlue, je třeba nainstalovat Asterisk. Vyžadována je minimálně verze 1.6. Asterisk nainstalujeme zadáním následující příkazu do terminálu:

```
sudo apt-get install asterisk
```

Následujícím příkazem provedeme instalaci balíků, na kterých VoiceGlue závisí:

```
sudo apt-get install libxerces-c2-dev xulrunner-dev flite
curl libssl-dev libbsd-resource-perl libmodule-build-perl
```



Obrázek 14: Registrace nové VXML aplikace

Nyní se dostáváme k instalaci VoiceGlue. Prvním krokem je získání zdrojových kódu. Zdrojové kódy stáhneme zadáním příkazu:

```
wget "http://github.com/voiceglue/voiceglue/tarball/0.14"
```

Po stažení bychom měli mít dispozici archiv obsahující zdrojové kódy. Následující příkazy provedou rozbalení archivu, přepnutí do složky s rozbaleným obsahem archivu a spuštění instalačního skriptu:

- `tar -xzf 0.14`
- `cd voiceglue-voiceglue-b12c2a2`
- `sudo ./doc/install-voiceglue`

Pokud instalace proběhla úspěšně, měli bychom mít nainstalováné následující tři nové služby:

- dynlog
- phoneglue
- voiceglue

Pokud si přejeme, aby se tyto služby spouštěly automaticky po startu systému, můžeme tak učinit následujícím příkazem:

```
for i in dynlog phoneglue voiceglue; do update-rc.d $i
defaults; done
```

Nainstalované služby zatím spouštět nebudeme.

7.3.2 Konfigurace

Pokud máme vše úspěšně nainstalováno, můžeme přejít ke konfiguraci. Nejdříve je nutné nakonfigurovat službu *phoneglue*. Prostřednictvím této služby probíhá komunikace mezi Asteriskem a VoiceGlue. Potřebnou konfiguraci provedeme editací souboru `/etc/asterisk/manager.conf`. Výpis 21 obsahuje kód, který je potřeba přidat na konec tohoto souboru.

```
1 [phoneglue]
2 secret=phoneglue
3 read = system,call,log,verbose,command,agent,user,originate
4 write = system,call,log,verbose,command,agent,user,
    ↪ originate
```

Výpis 21: Konfigurace souboru `/etc/asterisk/manager.conf`

Dále je nutné se ujistit, že je v tomto souboru obsažen řádek obsahující `enabled=yes`.

Dalším krokem je vytvoření SIP účtu. Do souboru `/etc/asterisk/sip.conf` přídáme kód obsažený ve výpise 22. Jméno uživatele a heslo si samozřejmě můžeme zvolit libovolně.

```
1 [ondrej]
2 type=friend
3 secret=heslo
4 userid=ondrej
5 host=dynamic
6 context=phoneglue
```

Výpis 22: Konfigurace souboru `/etc/asterisk/sip.conf`

Nyní je potřeba nakonfigurovat přepojování hovorů z Asterisku do VoiceGlue. Tuto konfiguraci provedeme v souboru `/etc/asterisk/extensions.conf`. Do souboru přidáme kód, který je obsažen ve výpise 23.

```
1 [phoneglue]
2 exten => dp,1,Answer
3 exten => dp,n,Set(vxmlurl=http%3A%2F%2Flocalhost%3A9000%2
    ↪ Fvxml)
4 exten => dp,n,Set(sessionid=${RAND(1,9999)})
5 exten => dp,n,Agi(agi://localhost/url=${vxmlurl}%3Fsession.
    ↪ sessionid=${sessionid})
6 exten => dp,n,Hangup
```

Výpis 23: Konfigurace souboru `/etc/asterisk/extensions.conf`

Kód vytváří nový kontext `[phoneglue]`. Tento kontext slouží k obsluze hovorů, které mají být zpracovány pomocí VoiceGlue. Kód dále registruje nové rozšíření představující naši hlasovou aplikaci. Adresa tohoto rozšíření je ”dp”.

Příkaz uvedený na druhém řádku provede příjem hovoru.

Třetí řádek definuje proměnnou `vxmlurl` obsahující URL adresu, na které běží náš hlasový portál. Hodnota je zakódována do takové podoby, aby ji bylo možné bezpečně přenést parametrem URL adresy.

Čtvrtý řádek definuje proměnnou `sessionid` a nastavuje její hodnotu na náhodné číslo. Hodnota této proměnné slouží jako jedinečný identifikátor hovoru (viz 6.2.2.1).

Pátý řádek provede prostřednictvím AGI rozhraní přepojení hovoru do subsystému VoiceGlue.

Šestý řádek provede zavěšení hovoru.

Nyní bychom měli mít vše správně nastaveno a můžeme tedy provést zkušební hovor na SIP adresu `dp@localhost`. Hovor můžeme vytočit z jakéhokoliv SIP klienta, např. Ekiga.

8 Zátěžové testy portálu

Tato kapitola je věnována zátěžovým testům hlasového portálu, jehož návrh byl popsán v předchozích kapitolách. Vzhledem k licenčním omezením[18], které jsou spojeny s používáním IVR Voxeo Prophecy nebylo bohužel možné nad touto platformou zátěžové testy provést. Proto bude předmětem našich testů pouze hlasová platforma VoiceGlue.

Zátěžové testy budou spouštěny na virtuálním stroji, který je poháněn jednojádrovým procesorem o taktu 2GHz a má k dispozici 2GB operační paměti. Jako operační systém je použita serverová varianta Ubuntu 10.04.4 LTS. Na tomto stroji je nainstalována softwarová ústředna Asterisk ve verzi 1.6.2.5. Nad touto ústřednou je spuštěna hlasová platforma VoiceGlue ve verzi 0.14.

Předmětem zátěžových testů hlasového portálu je analýza výpočetní a paměťové náročnosti obsluhy různého počtu připojených klientů. V testech je měřeno celkové vytížení systému i vytíženost konkrétních procesů. Zátěžové testy jsou postaveny tak, aby bylo možné zároveň analyzovat využití hlasové platformy i aplikačního serveru. Pro tuto analýzu byly zvoleny následující procesy:

- asterisk
- voiceglue
- phoneglue
- node
- mongod

Analýza vytíženosti procesů $asterisk$, $voiceglue$ a $phoneglue$ nám poskytne informaci o výpočetní a paměťové náročnosti běhového prostředí hlasové platformy. Z vytíženosti procesů $node$ a $mongod$ budeme zjišťovat výpočetní a paměťovou náročnost aplikačního serveru, konkrétně webového a databázového serveru.

8.1 Scénář zátěžových testů

Zátěžové testy spočívaly ve vytvoření zátěže na hlasovém portálu simulací n připojených klientů. Každý připojený klient inicioval SIP hovor. Schéma tohoto hovoru je zobrazeno na obrázku 15.

Hovor sestává z posloupnosti následujících akcí:

1. iniciace hovoru SIP zprávou INVITE
2. přehrání uvítací zprávy a výzvy k zadání přihl. jména
3. zadání přihl. jména ”1” prostřednictvím sekvence DTMF voleb 1 a #
4. přehrání výzvy k potvrzení přihl. jména

[18] Verze, která je dostupná zdarma je omezena pouze na dva porty, čili na maximálně dva souběžné hovory.

Celkový počet hovorů	Počet souběžných hovorů
5	5
10	5
10	10
20	10
20	20
40	20
40	40
80	40
60	60
120	60

Table 5: Varianty provedených zátěžových testů

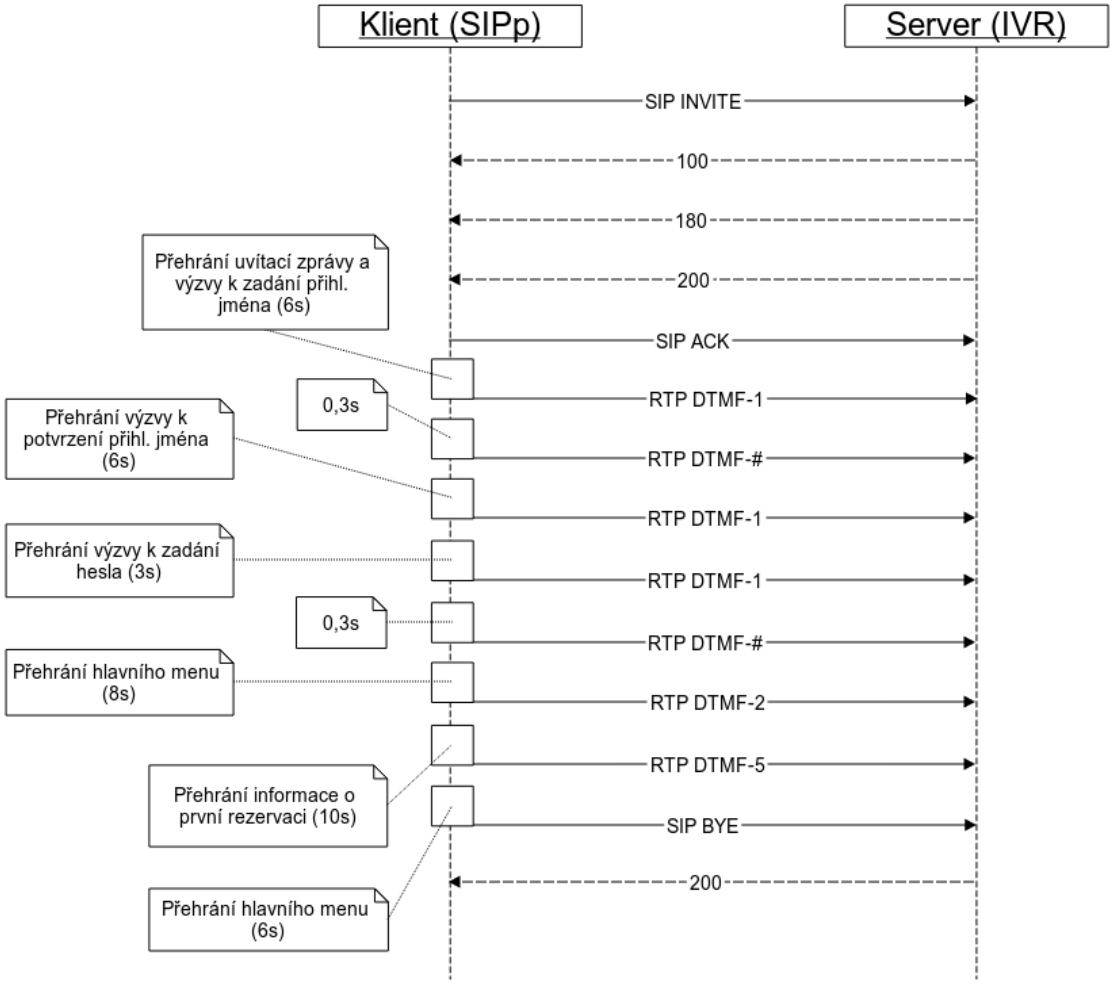
5. potvrzení zadaného přihl. jména prostřednictvím DTMF volby 1
6. přehrání výzvy k zadání přihl. hesla
7. zadání přihl. hesla "1" prostřednictvím sekvence DTMF voleb 1 a #
8. přehrání nabídky hlavního menu
9. zvolení "Seznamu rezervací" prostřednictvím DTMF volby 2
10. přehrání informací o první aktivní rezervaci
11. návrat do hlavního menu prostřednictvím DTMF volby 5
12. přehrání nabídky hlavního menu
13. ukončení hovoru SIP zprávou BYE

Tento hovor byl poté spouštěn vůči hlasovému portálu. Hovory byly spouštěny v různém počtu instancí. V každém testu byl zároveň spouštěn různý počet souběžných instancí hovoru. V každém testu bylo měřeno procesorové a paměťové vytížení systému a klíčových procesů. Kapitola 8.3 je věnována výsledkům měření.

V tabulce 5 jsou uvedeny všechny varianty provedených zátěžových testů.

8.2 Použité nástroje

Pro provedení zátěžových testů bylo potřeba zvolit vhodné nástroje. V první řadě bylo nutné použít nástroj umožnující simulaci klientů, kteří se připojují k hlasovému portálu. Dále bylo potřeba nástroje, který nám umožní měřit celkové vytížení systému a vytížení konkrétních procesů.



Obrázek 15: Schéma simulovaného hovoru

8.2.1 SIPp

SIPp je nástroj, který umožňuje generovat síťový provoz spojený se SIP protokolem. Tento nástroj umí ze XML souboru načíst definici SIP dialogu a na základě této definice generovat SIP zprávy. SIPp také umožňuje generovat pakety RTP protokolu. Podpora RTP protokolu je v testovacím scénáři využita pro přenos DTMF volby. Součástí instalace tohoto nástroje jsou k dispozici zachycené RTP pakety jednotlivých DTMF voleb ve formátu pcap.

Nástroj při spuštění přebírá několik nastavení, kterými je možné řídit např. celkový počet SIP spojení, které se mají vykonat nebo maximální počet souběžných SIP spojení.

Pro naše účely byl nástroj spouštěn např. následujícím příkazem:

```
sudo sipp -sf scenario.xml -s dp -m 6 -l 2 -i
192.168.1.1 -nd -trace_stat -fd 250 192.168.1.2
```

SIPp je nutné spouštět pomocí příkazu `sudo` a to z toho důvodu, aby program mohl vytvořit sokety, které jsou nutné pro odesílání RTP paketů.

Přepínač `sf` udává cestu k XML souboru, ve kterém je obsažena definice SIP dialogu - testovacího scénáře.

Pomocí přepínače `s` specifikujeme název vytáčené SIP služby.

Přepínačem `m` nastavujeme celkový počet vykonaných SIP spojení (hovorů).

Přepínačem `l` specifikujeme maximální počet souběžně vykonávaných SIP spojení (hovorů).

Pomocí přepínače `i` specifikujeme zdrojovou IP adresu. Tuto informaci je nutné uvést v případě, kdy stroj, na kterém SIPp běží, obsahuje více síťových rozhraní.

Uvedením přepínače `nd` vypínáme výchozí chování programu, které spočívá v ukončení běhu programu při příjmu neočekávané SIP zprávy.

Přepínač `trace_stat` zapíná logování statistik o průběhu jednotlivých spojení.

Poslední údaj, který je nutné uvést, je IP adresa stroje, na kterém běží hlasový portál.

Tento nástroj byl pro testy zvolen hlavně proto, že umožňuje jednoduchým způsobem nasimulovat n připojených klientů.

SIPp byl během zátěžových testů spouštěn na samostatném stroji, aby nedocházelo ke zvýšení vytíženosti měřeného systému vlivem běhu tohoto programu.

8.2.2 top

top je nástroj, který je součástí UNIX-like operačních systému a slouží k monitorování aktuálně běžících procesů a podprocesů. Tento program umožňuje periodicky zjišťovat informace o celkovém využití CPU a paměti. Zároveň poskytuje informace o využití systémových prostředků aktuálně běžícími procesy v systému.

Pro účely našich zátěžových testů byl program spouštěn např. následujícím příkazem:

```
top -b -d 100 > out.txt
```

Přepínač `b` spouští program v dávkovém módu. Program je nutné v tomto módu spustit v případě, že chceme přesměrovat výstup do souboru.

Pomocí přepínače `d` specifikujeme dobu, po jejichž uplynutí program vygeneruje nový snímek vytíženosti systému. Ve výše uvedeném případě se tak děje každých 100ms.

Tento nástroj byl při běhu zátěžových testů spouštěn na stroji, na kterém byla spuštěna instance hlasové platformy a aplikačního serveru.

8.2.3 Automatizační skript

Výše jsem popsal dva nástroje, které byly využity pro simulaci připojovaných klientů a měření vytíženosti systému. Zátěžové testy byly postupně spouštěny s různými parametry. Pro zjednodušení spouštění zátěžových testů byl vytvořen *bash* skript, kterému je možné při spouštění předat tři parametry. Ve výpise 24 je uveden zdrojový kód tohoto skriptu.

```
1  #!/bin/bash
2
3  outDir=out
4  outFile=./$outDir/`date "+%H_%M_%S"`
5  capture_delay=$1
6  total_calls=$2
7  concurrent_calls=$3
8  local_ip=192.168.1.100
9  remote_ip=192.168.1.112
10
11 mkdir $outDir 2> /dev/null
12 sshpass -p '12345' ssh ondrej@$remote_ip "top -b -d ${
       ↪ capture_delay}" > $outFile.temp &
13 sudo sipp -sf scenario.xml -s dp $remote_ip -m $total_calls
       ↪  -l $concurrent_calls -i $local_ip -nd -trace_stat -fd
       ↪  $capture_delay 2> ./$outDir/err.log
14 sshpass -p '12345' ssh ondrej@$remote_ip "killall top"
15 ./parse.js $outFile.temp > $outFile.csv
16 rm $outFile.temp
```

Výpis 24: Zdrojový kód skriptu `start-bench.sh`

Tento skript je nutné spouštět z umístění, ve kterém je obsažen XML soubor s definicí testovacího scénáře. Umístění dále musí obsahovat adresář *pcap*, ve kterém jsou uloženy soubory se zaznamenanými RTP pakety DTMF voleb. Skript je poté možné spustit např. následujícím příkazem:

```
./start-bench.sh 0.2 20 10
```

První parametr udává periodu vzorkování vytížení systému. Hodnota se zadává ve vteřinách. Druhým parametrem udáváme celkový počet simulovaných hovorů, které chceme vyvolat. Posledním parametrem nastavujeme maximální počet souběžných hovorů.

Skript provede po spuštění následující operace:

1. vytvoření adresáře pro výstupní data
2. spuštění příkazu `top` na vzdáleném stroji a přesměrování výstupu na lokální stroj
3. spuštění příkazu `sipp` na lokálním stroji
4. ukončení příkazu `top` na vzdáleném stroji
5. spuštění skriptu, který transformuje výstup příkazu `top` do formátu CSV
6. smazání dočasného souboru

Je důležité poznamenat, že skript nezajišťuje spuštění hlasové platformy a aplikačního serveru na vzdáleném stroji. Tyto procesy je nutné spustit manuálně. Aplikační server je možné spustit následujícím příkazem:

```
PORT=9000 node --harmony index.js
```

Jedním z posledních operací, které skript provádí je spuštění skriptu `parse.js`. Tento skript slouží k transformaci zachyceného výstupu z programu `top` do formátu CSV, ze kterého je poté možné vygenerovat grafy. Ve zdrojovém kódu tohoto skriptu jsou nadefinována jména procesů, o nichž se mají zjišťovat informace o vytíženosti. Některé z těchto procesů mohou být spuštěny v několika instancích (samostatných procesech). Např. proces `asterisk` je obvykle spuštěn minimálně ve dvou instancích. V zachyceném výstupu z programu `top` je poté uveden zvlášť záznam o každé instanci. Skript tedy zajišťuje agregaci informace o vytíženosti dílčích procesů.

8.3 Výsledky měření

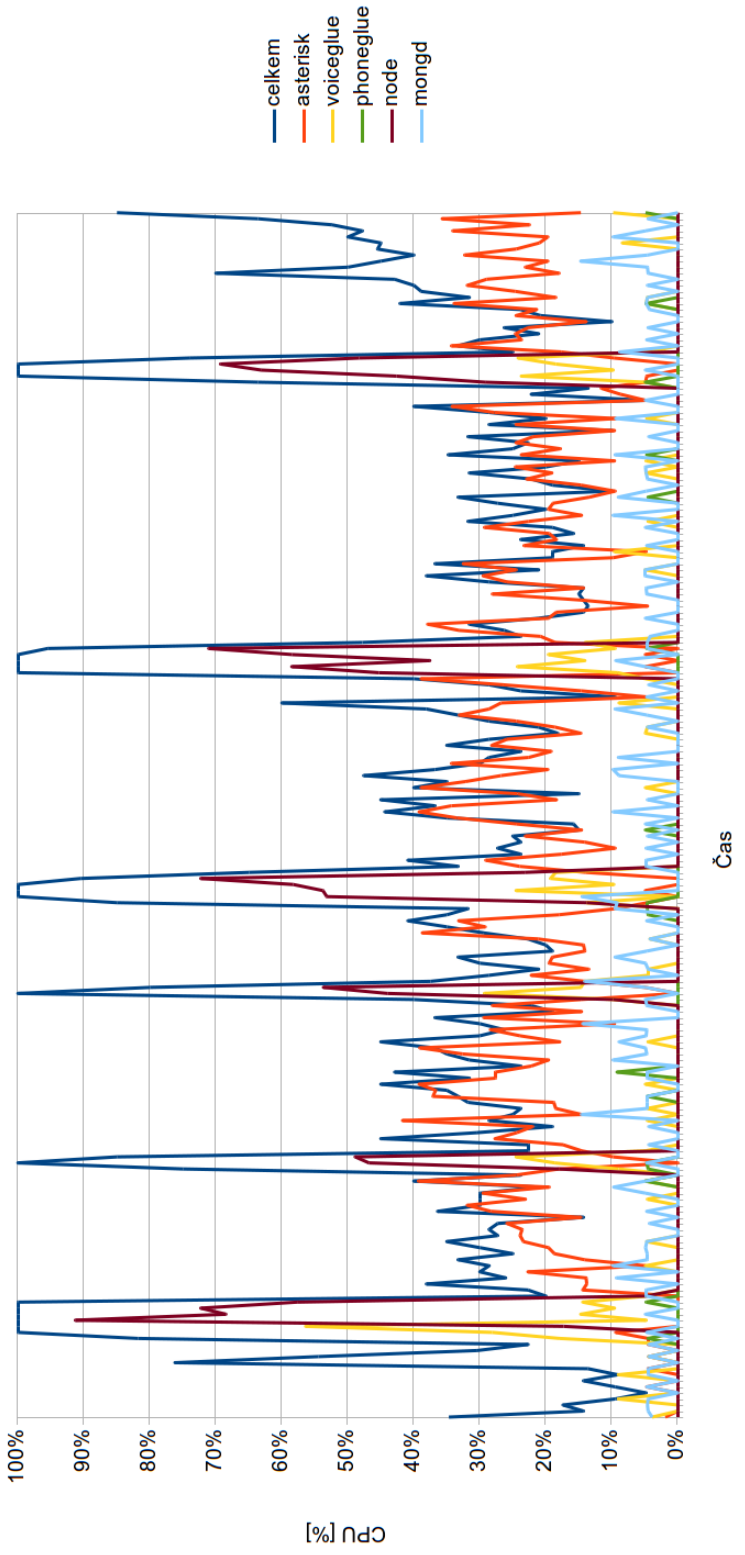
Měřením bylo zjištěno konkrétní vytížení procesoru a využití paměti RAM při různém počtu probíhajících hovorů v kombinaci s různým počtem souběžných hovorů (viz tabulka 5). V této kapitole uvedu grafy zachycující zmíněné využití hlasové platformy i aplikačního serveru. Nebudu zde uvádět grafy využití všech variant zátěžových testů, ale uvedu zde jen grafy nejzajímavější variant testů.

Ve všech následně uvedených grafech je zachyceno celkové využití procesoru nebo paměti RAM včetně podílu vytíženosti jednotlivých procesů. Na osu Y jsou vynášeny procentuální hodnoty využití konkrétního zdroje - procesoru nebo paměti RAM. Osa X reprezentuje časový průběh zátěžového testu. U osy X není uvedena žádná měrná jednotka. Není uvedena z toho důvodu, že nás nezajímá vytížení v konkrétním čase, spíše nás zajímá relativní distribuce vytíženosti zdroje v průběhu testu.

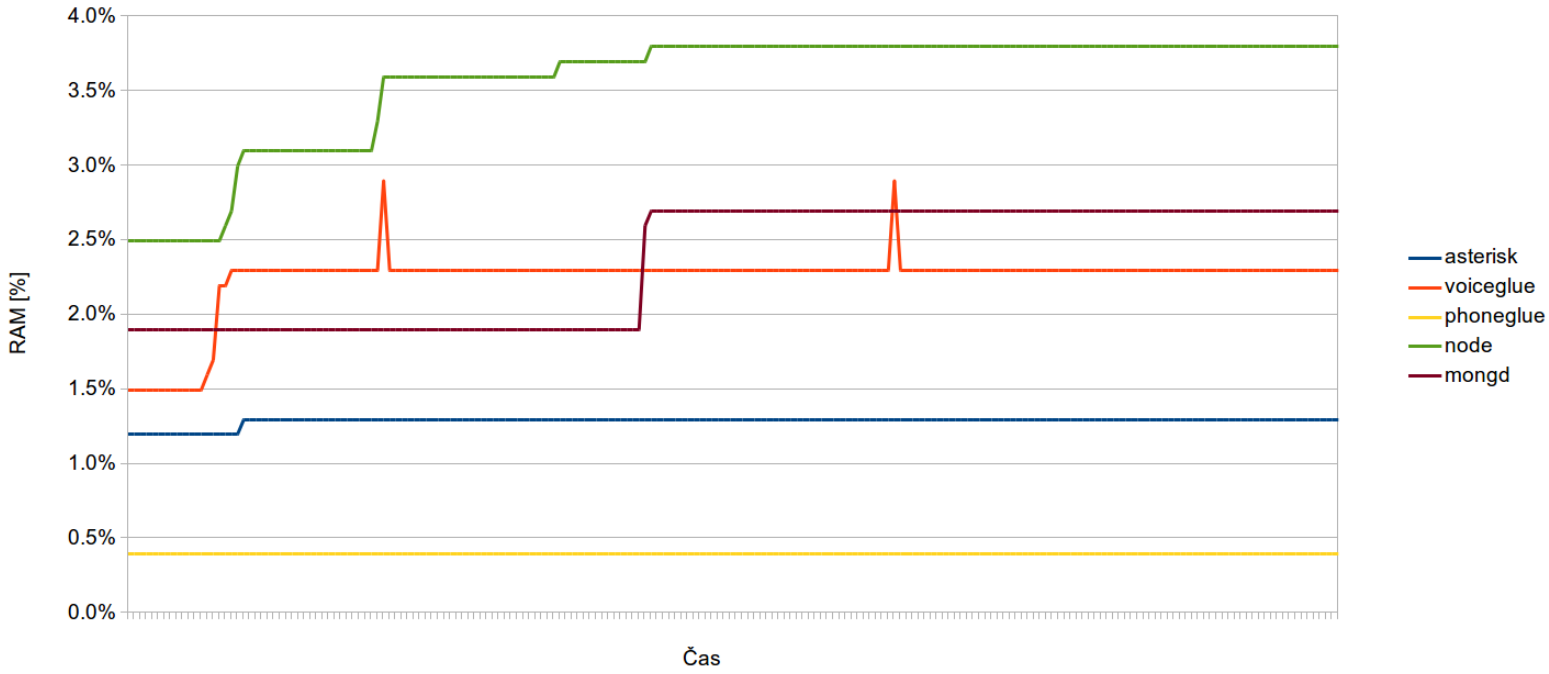
8.3.1 Zátěžový test č. 1

Konfigurace tohoto, nejméně náročného, zátěžového testu spočívala v uskutečnění celkového počtu 5 hovorů, kdy všech 5 hovorů bylo obslouženo souběžně.

Na obrázku 16 je zobrazen graf zachycující využití procesoru při této konfiguraci. Na obrázku 17 je zobrazeno využití paměti RAM.



Obrázek 16: Využití CPU při spuštění 5 souběžných hovorů, celkově 5 hovorů



Obrázek 17: Využití paměti RAM při spuštění 5 souběžných hovorů, celkově 5 hovorů

Z grafu vytíženosti procesoru je patrné, že celkové vytížení systému je maximální jen v určitých časových periodách, které nemají dlouhého trvání. Největší podíl na vytíženosti má proces $node$, který představuje aplikační server.

Procesy, které v testu zastupují hlasovou platformu, mají téměř konstantní podíl využití. Nejvíce vytíženým procesem hlasové platformy je proces $asterisk$, tedy hlasová ústředna.

Vytíženost paměti RAM nám prozrazuje, že téměř všechny procesy mají konstantní nároky na paměť v průběhu celého testu. Výjimkou je opět proces $node$, který představuje aplikační server. Právě u tohoto procesu je patrná postupná alokace paměti.

8.3.2 Zátěžový test č. 2

Konfigurace tohoto zátěžového testu spočívala v uskutečnění celkového počtu 20 hovorů. V tomto případě bylo souběžně obsluhováno vždy 10 hovorů. Celková doba trvání této varianty testu tedy zabrala dvojnásobný čas.

Tento typ testu nám může odhalit případné problémy s uvolňováním zdrojů mezi zpracováním jednotlivých "dávek" hovorů. Z grafů 18 a 19 však není žádná kumulace využití zdrojů patrná. Můžeme tedy usoudit, že zdroje jsou vždy správně uvolňovány.

Z vytíženosti procesoru (18) je patrné, že zpracování hovorů v této konfiguraci systém stále zvládá bez sebemenšího problému - stále není ani zdaleka využit veškerý procesorový čas.

V grafu vytíženosti paměti RAM (19) si můžeme všimnout zvýšeného podílu na využití paměti u procesu $voiceglue$. U paměťové vytíženosti ostatních procesů nedošlo k výraznějšímu nárůstu.

8.3.3 Zátěžový test č. 3

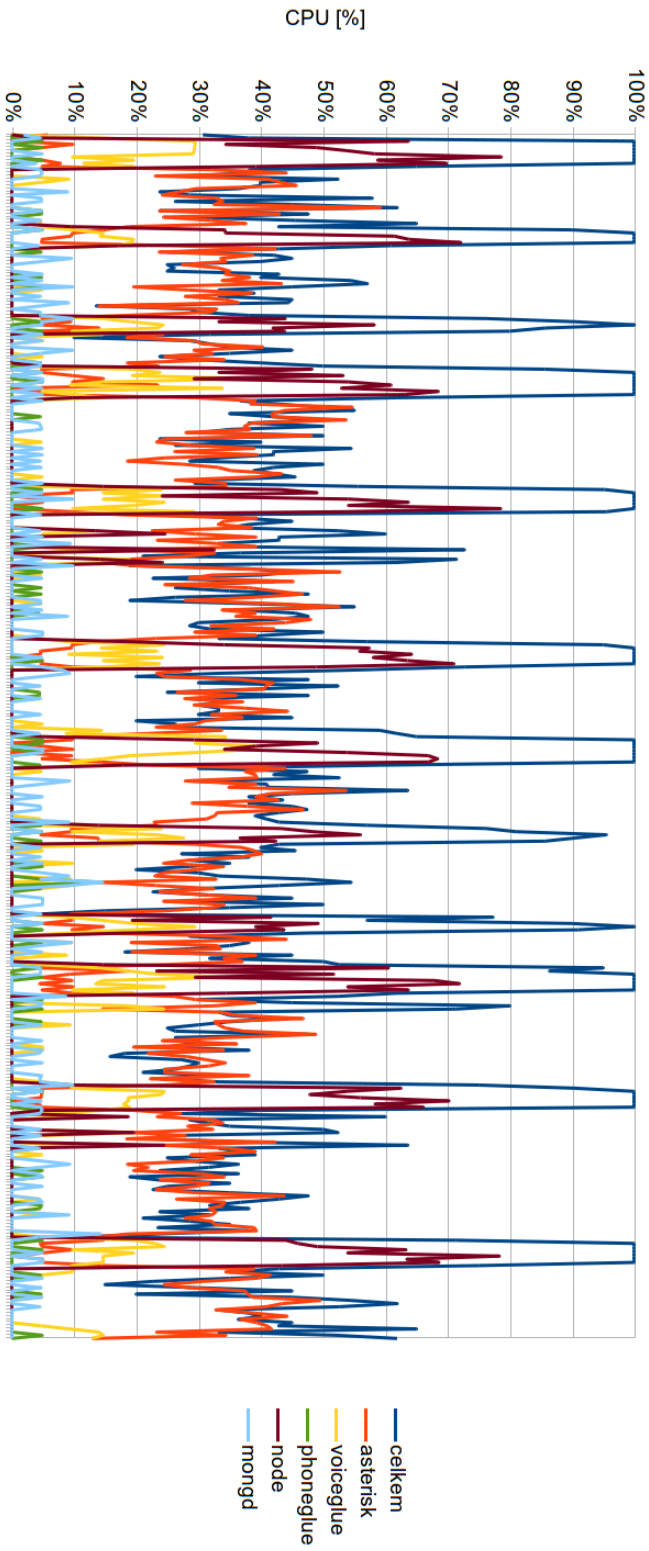
Konfigurace této varianty zátěžového testu spočívala v uskutečnění celkového počtu 60 hovorů, kdy všech 60 hovorů bylo obslouženo souběžně.

Využití procesoru (20) je u této varianty testu téměř stoprocentní po celou dobu trvání zátěžového testu. Je tedy patrné, že pro danou hardwarovou konfiguraci stroje (viz 8) jsme se přiblížili k maximálnímu počtu hovorů, které je možné reálně obsloužit. Při větším počtu souběžných hovorů by již mohlo docházet k nepřijatelným prodlevám mezi iniciací hovoru a jeho obsloužením. Tyto prodlevy by dále mohly způsobovat problémy, které by např. mohly vést k nutnosti přeposílat některé datové pakety.

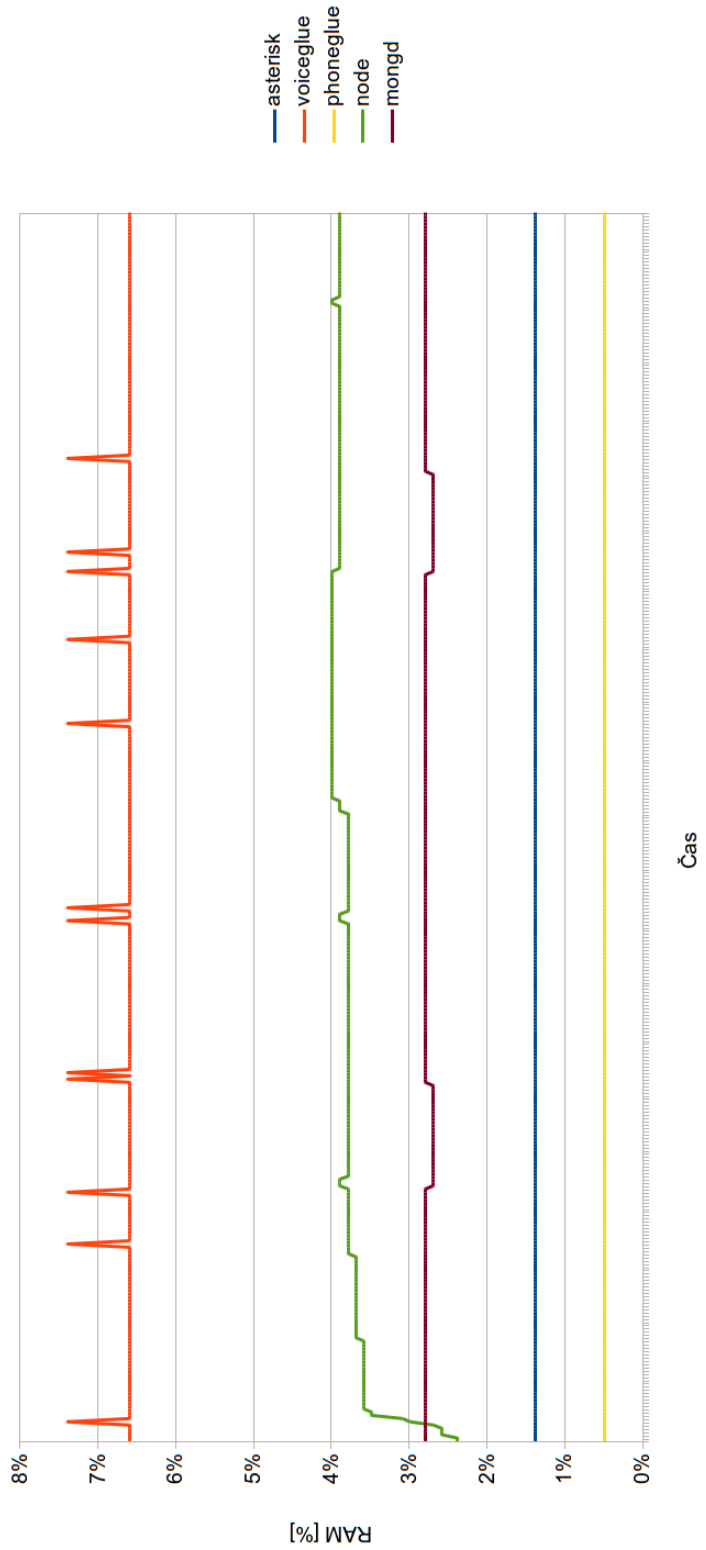
Využití paměti RAM, které můžeme vidět v grafu na obrázku 21, opět ukazuje mírné zvýšení nároků na paměť u procesu $voiceglue$. U ostatních procesů není patrná žádná výrazná změna oproti výsledkům z předchozího testu.

8.3.4 Obecné zhodnocení

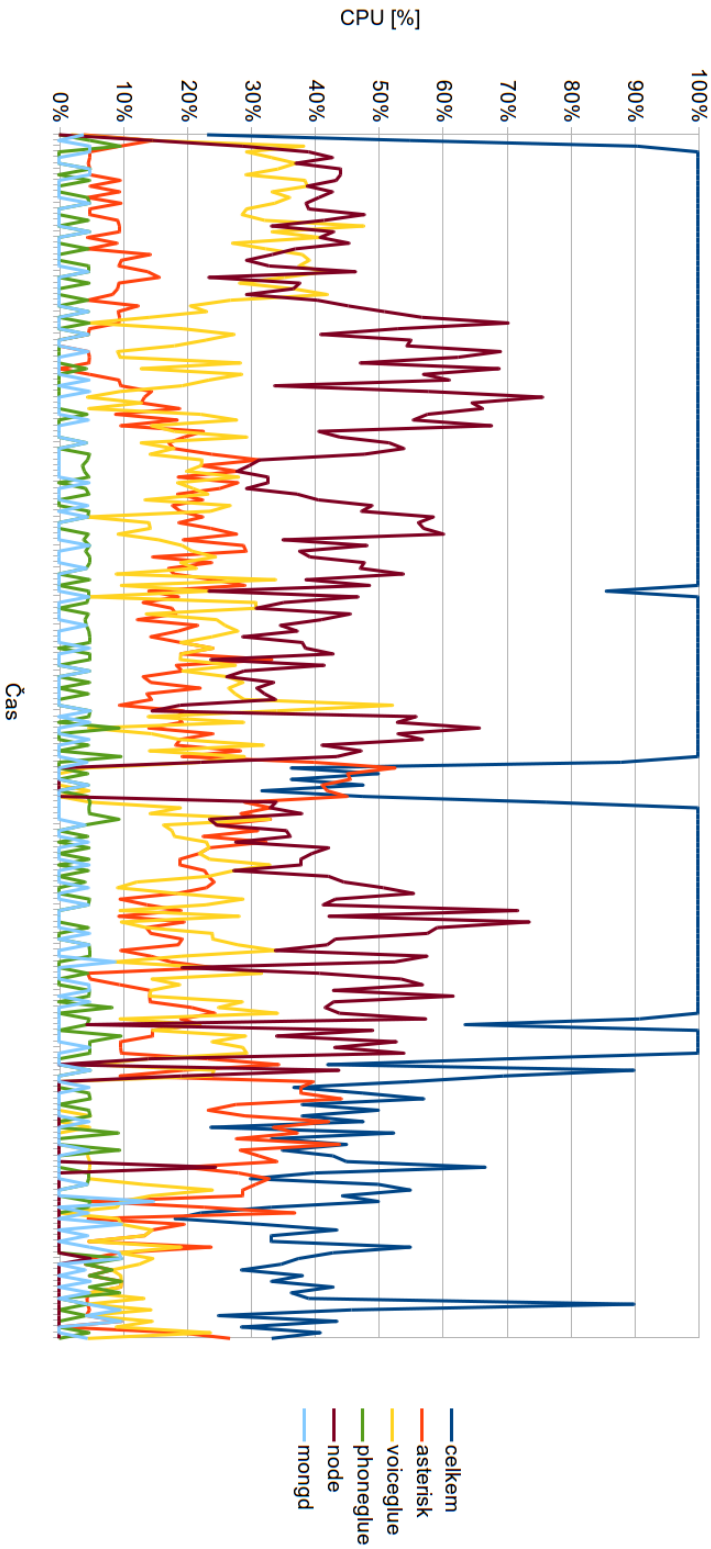
Výsledky jednotlivých variant zátěžových testů můžeme obecně shrnout následovně.



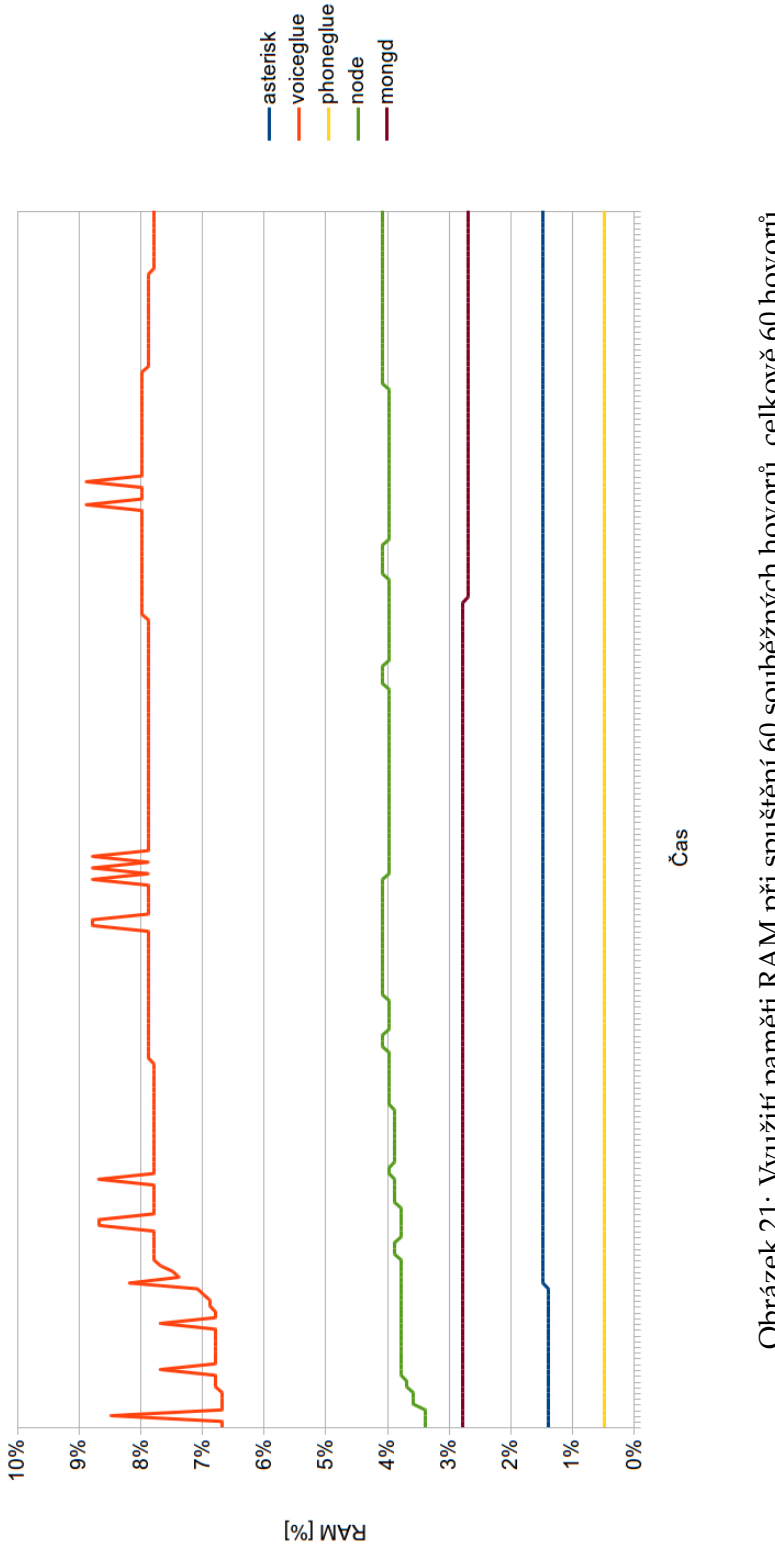
Obrázek 18: Využití CPU při spuštění 10 souběžných hovorů, celkově 20 hovorů



Obrázek 19: Využití paměti RAM při spuštění 10 souběžných hovorů, celkově 20 hovorů



Obrázek 20: Využití CPU při spuštění 60 souběžných hovorů, celkově 60 hovorů



Obrázek 21: Využití paměti RAM při spuštění 60 souběžných hovorů, celkově 60 hovorů

Měřením využití procesoru byl přibližně zjištěn maximální počet hovorů, které lze na dané hardwarové platformě (1 x CPU@2GHz a 2GB RAM) obsloužit souběžně, aniž by začalo docházet k nárůstu doby potřebné pro obsloužení hovoru.

Z analýzy využití paměti RAM naopak vyplynulo, že zde existují ještě velké rezervy. Využití paměti monitorovaných procesů se nezvýšilo na nikterak vysokou hodnotu. Využití paměti jednotlivými procesy bylo téměř ve všech variantách zátěžových testů konstantní. Využití paměti bylo téměř stejné i napříč různými testy. Pouze u procesu $voiceglue$ byl mezi testy patrný mírný nárůst nároku na paměť. I přes tento mírný postupný nárůst se využití paměti tímto procesem pohybovalo pouze mezi 2-9%.

Pro obsluhu většího počtu souběžných hovorů by bylo nutné zajistit vyšší výpočetní výkon, resp. procesor s více jádry. Další možností, jak zajistit vyšší škálovatelnost a stabilitu celého systému, je provozovat jednotlivé komponenty na různých serverech. Mohli bychom tedy například provozovat aplikační a hlasový server na odlehlých serverech.

9 Závěr

Diplomová práce se zabývala návrhem hlasového portálu na platformě VoiceXML.

Úvodní kapitola nejdříve představila jazyk VoiceXML a jeho architekturu, strukturu VoiceXML dokumentů a seznámila nás se základy návrhu hlasových aplikací pomocí jazyka VoiceXML.

Text kapitoly dále pokračoval podrobným představením dvou hlasových platforem VoiceXML. Konkrétně zde byla představena architektura a vlastnosti komerční platformy Voxeo Prophecy a opensource platformy VoiceGlue.

Následující dvě kapitoly se zabývaly popisem technologií, které úzce souvisí s VoiceXML, resp. s návrhem hlasových portálů. Byla zde představena technologie pro syntézu řeči (TTS) a technologie pro automatické rozpoznávání řeči (ASR).

Hlavní náplň práce je obsažena v páté a šesté kapitole. Na začátku páté kapitoly je představen informační systém pro rezervaci letenek. Následující text kapitoly popisuje návrh webového rozhraní tohoto informačního systému. Text se dále zabývá podrobnějším představením technologií, které byly pro návrh webového portálu zvoleny.

Náplní šesté kapitoly je popis návrhu a následné implementace hlasového portálu, který doplňuje webové rozhraní informačního systému o možnost hlasového ovládání prostřednictvím samoobslužného hlasového kanálu. Hlasový portál je možné ovládat pomocí DTMF volby. Ovládání pomocí DMTF volby bylo zvoleno z důvodu možnosti provozovat hlasový portál i na platformách, které nedisponují možností automatického rozpoznávání řeči (ASR).

Kapitola je rozdělena na dvě hlavní části. První část se zabývá návrhem knihovny, která má za úkol zpřehlednit a urychlit vývoj hlasových aplikací na platformě VoiceXML. Na tuto část navazuje část, kde je představen návrh konkrétních částí hlasového portálu pomocí vytvořené knihovny.

Sedmá kapitola popisuje konfiguraci, kterou je nutné provést pro uvedení hlasového portálu do provozu. Je zde popsána konfigurace pro hlasového platformy Voxeo Prophecy a VoiceGlue.

Poslední kapitola se zabývá zátěžovými testy hlasového portálu. Cílem zátěžových testů byla analýza využití systémových zdrojů a zjištění reálných hardwarových a softwarových požadavků na provozní prostředí hlasového portálu.

Na začátku kapitoly byla nejdříve představena konfigurace testovacího prostředí. Následovalo představení testovacího scénáře. Zbytek kapitoly byl věnován rozboru provedených testů.

Jako největší přínos této práce považuji vytvoření knihovny sloužící k usnadnění a urychlení vývoje hlasových aplikací na platformě VoiceXML.

Bc. Ondřej Ždych

10 Použité zdroje

[1] Shukla, Charul; Dass, Avnish; Gupta, Vikas, *VoiceXML 2.0 Developer's Guide : Building Professional Voice-enabled Applications with JSP, ASP Coldfusion*, Dream Tech Software India Inc., 2002.

[2] Peter Trilec, *Porovnání volně dostupných VoiceXML interpreterů, bakalářská práce*, Masarykova univerzita, 2008.

[3] Digium, Inc., *Web projektu Asterisk*, http://www.asterisk.org/get-started, 2014.

[4] VoiceGlue, *Web projektu VoiceGlue*, http://www.voiceglue.org, 2014.

[5] VoiceGlue, *Architektura VoiceGlue*, https://github.com/voiceglue/voiceglue/wiki/Features-and-Architecture, 2014.

[6] Wikipedia, *Extensible Markup Language*, http://cs.wikipedia.org/wiki/Extensible_Markup_Language, 2014.

[7] Pavel Bezpalec; Katedra telekomunikační techniky, ČVUT Praha, *Sítě integrovaných služeb*, http://www.comtel.cz/files/download.php?id=5459, 2010.

[8] Linguatec Language Technologies, *Technologie převádění textu na mluvené slovo (Text-To-Speech)*, http://www.linguatec.cz/products/tts/information/technology, 2014.

[9] Wikipedia, *Speech synthesis*, http://en.wikipedia.org/wiki/Speech_synthesis, 2014.

[10] Alan W. Black, *Flite: a small, fast run time synthesis engine*, http://www.festvox.org/flite, 2014.

[11] Wikipedia, *Festival Speech Synthesis System*, http://en.wikipedia.org/wiki/Festival_Speech_Synthesis_System, 2014.

[12] Wikipedia, *Festival Speech Synthesis System*, http://en.wikipedia.org/wiki/Cepstral_(company), 2014.

[13] Cepstral LLC, *Cepstral TTS*, https://www.cepstral.com, 2013.

[14] Wikipedia, *Speech recognition*, http://en.wikipedia.org/wiki/Speech_recognition, 2014.

[15] Loquedo, *Loquendo ASR*, http://www.sintel.com/bibli/telechargement/168/document_Multi.pdf, 2010.

[16] Technická univerzita v Liberci, *Laboratoř počítačového zpracování řeči*, https://www.ite.tul.cz/speechlab, 2014.

[17] Nuance Communications, Inc., *Web společnosti Nuance*, http://www.nuance.com, 2014.

[18] Google, Inc., *Dokumentace k frameworku AngularJS*, http://docs.angularjs.org, 2014.

[19] MongoDB, Inc., *Web projektu MongoDB*, http://www.mongodb.org, 2013.

[20] LearnBoost, Inc., *Web projektu Mongoose*, http://www.mongoosejs.com, 2011.

[21] Joyent, Inc., *Web platformy Node.js*, http://www.nodejs.org, 2014.

[22] TJ Holowaychuk, *Web frameworku Express.js*, http://www.expressjs.com, 2014.

[23] Opensource komunita, *Web věnovaný nástroji Grunt*, http://gruntjs.com, 2014.

[24] Voxeo, *VXML dokumentace*, http://www.vxml.org, 2013.

[25] MongoDB, Inc., *Popis instalace MongoDB*, http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu, 2014.

[26] Joyent, Inc., *Instalační balíky Node.js*, http://nodejs.org/download, 2014.

A Přílohy

A.1 Obsah CD

- diplomová práce ve formátu PDF
- zdrojové kódy webového portálu
- zdrojové kódy hlasového portálu
- zdrojové kódy VXML knihovny
- doprovodné skripty

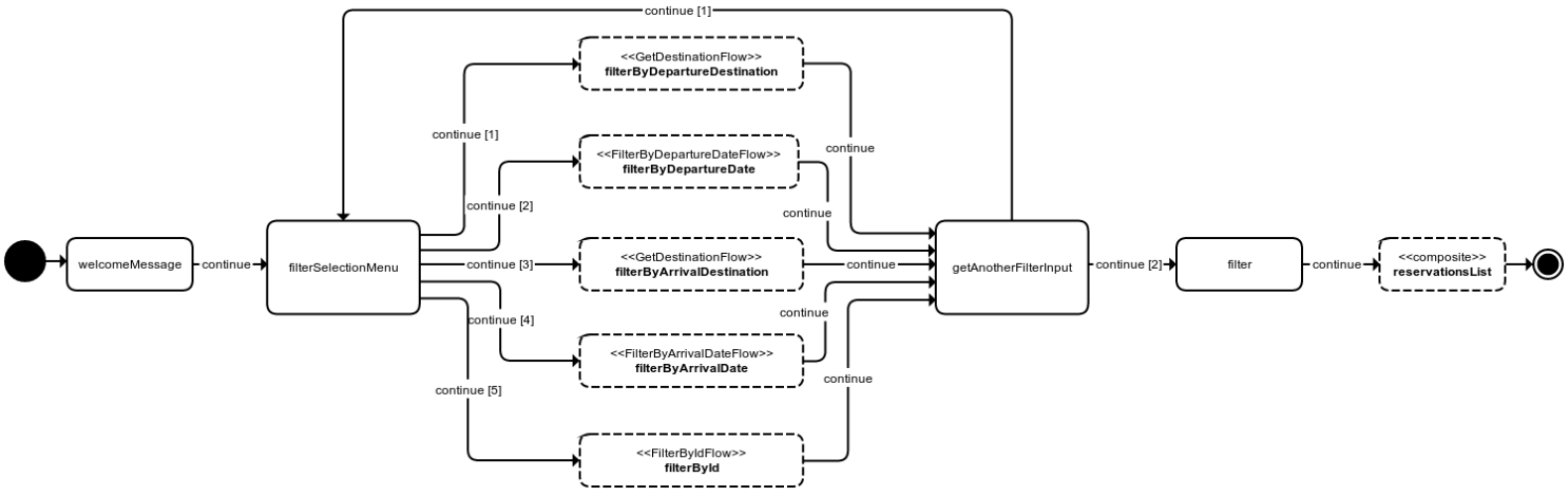
A.2 Diagramy jednotlivých částí hlasového portálu

V této příloze se nachází diagramy všech částí hlasového portálu. Pro přehlednost zde uvádím odkazy na diagramy částí, které byly uvedeny přímo v textu práce.

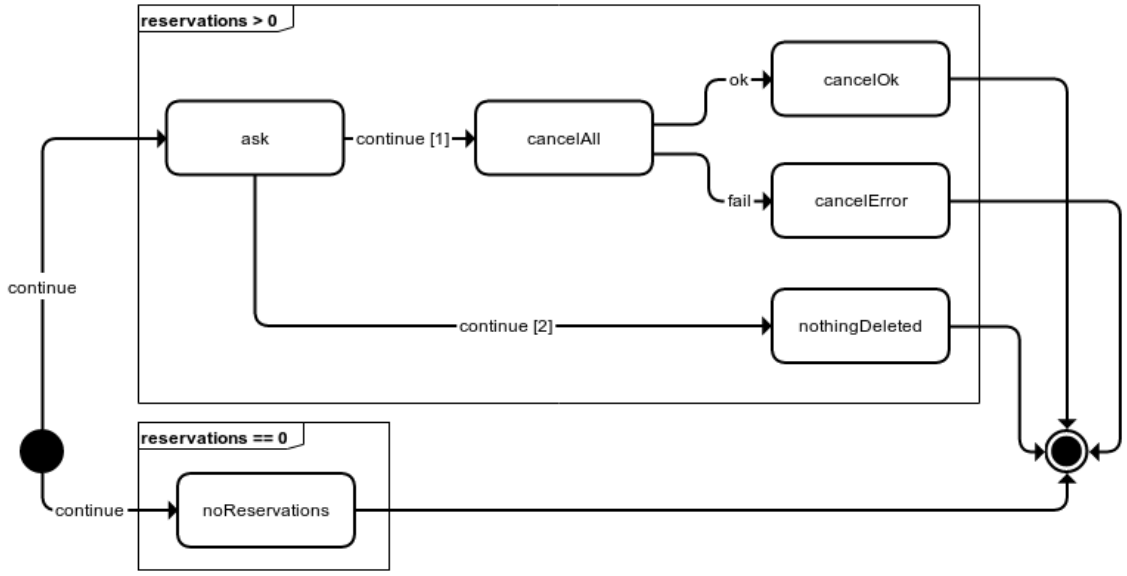
A.2.1 V textu uvedené diagramy

- Hlavní struktura hlasového portálu je znázorněna na obrázku 8.
- Vnitřní struktura stavu $getLoginData$ je znázorněna na obrázku 9.
- Struktura komponenty pro zadání data je znázorněna na obrázku 10.
- Struktura komponenty pro zadání textového vstupu je znázorněna na obrázku 11.
- Struktura komponenty pro zvolení destinace je znázorněna na obrázku 12.
- Vnitřní struktura stavu $destinationSelection$ komponenty pro zvolení destinace je znázorněna na obrázku 13.

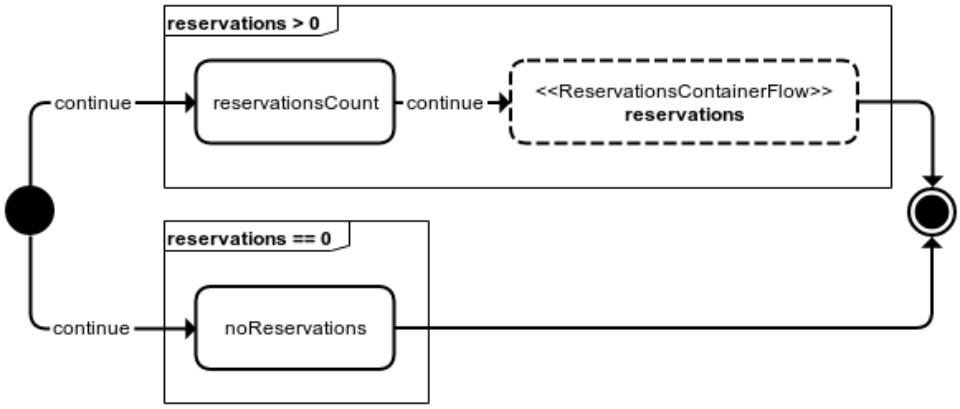
A.2.2 V textu neuvedené diagramy



Obrázek 22: Struktura stavu $createNew$ ($CreateNewFlow$)

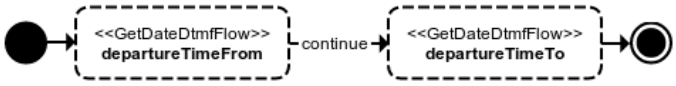


Obrázek 23: Struktura stavu $cancelActive$ ($CancelActiveFlow$)

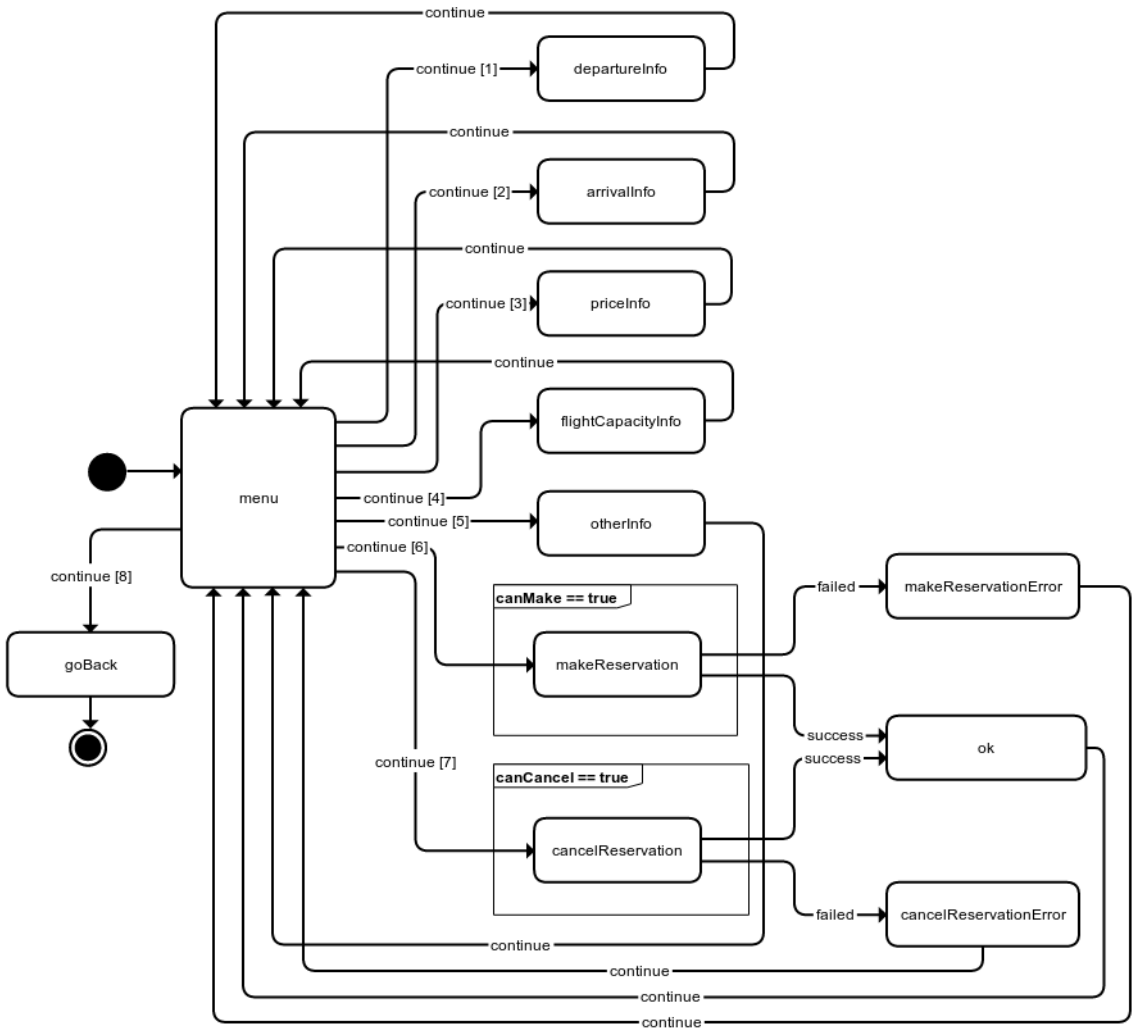


Obrázek 24: Struktura stavu $listActive$ ($ListActiveFlow$)

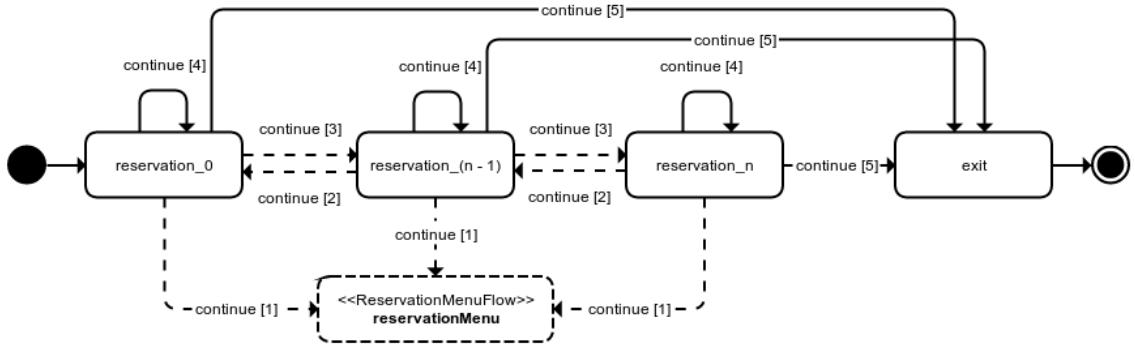
Obrázek 25: Struktura stavu $filterByArrivalDate$ ($FilterByArrivalDateFlow$)



Obrázek 26: Struktura stavu $filterByDepartureDate$ ($FilterByDepartureDateFlow$)



Obrázek 27: Struktura stavu $reservationMenu$ ($ReservationMenuFlow$)



Obrázek 28: Struktura stavu $reservationsContainer$ ($ReservationsContainerFlow$)