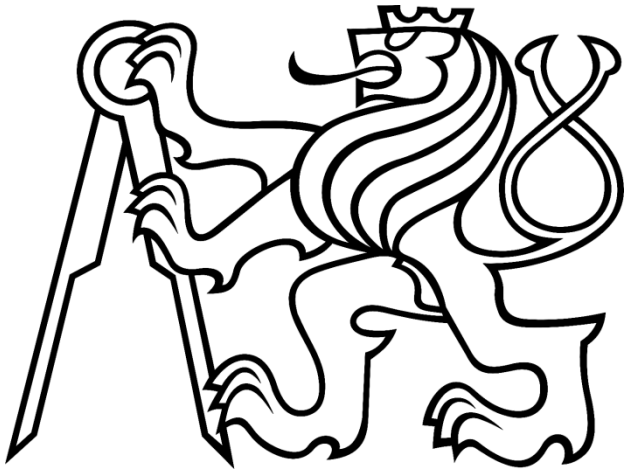
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Plánování, vytváření a obsluhování telefonních hovorů

Oscar Hernández

Vedoucí práce: Ing. Mannová Božena, Ph.D.

Studijní program: Softwarové technologie a management, bakalářský

Obor: Softwarové inženýrství

22. května 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Oscar Hernández**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Plánování, vytváření a obsluhování telefonních hovorů**

Pokyny pro vypracování:

Cílem práce je vytvořit aplikaci automatizující plánování, vytváření a obsluhování telefonních hovorů v propojení s VoIP (Voice over IP) serverem.
Aplikace v pravidelných intervalech kontroluje databázi, která obsahuje požadované hovory. Nastane-li čas, kdy má hovor proběhnout, aplikace pomocí VoiP serveru vytočí požadované číslo a po zvednutí volaného telefonu přehraje sadu nahrávek specifikovanou v databázi. Po ukončení hovoru je výsledek zaznamenán do databáze(hovor proběhl , délka hovoru, volané číslo bylo obsazené, nedostupné). Aplikace realizuje i posílání jednoduchých příkazů (např. jednorázové vytvoření hovoru, vypnutí nebo restart aplikace).
K vypracování použijte programovací jazyk Java, technologii Maven pro řízení závislostí a Open source VoIP server Asterisk.
Plánování, vytváření a obsluhování telefonních hovorů

Seznam odborné literatury:

Pressman, R.S, Maxim, B.: Software Engineering a Practitioner's Approach, McGraw-Hill, 2014, ISBN-13: 978-0078022128 ISBN-10: 0078022126
Sonatype Company: Maven: The Definitive Guide, O'REILLY, 2008, ISBN-13: 978-0596517335 ISBN-10: 0596517335

Vedoucí: Ing. Božena Mannová, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 17. 4. 2015

Poděkování

Děkuji vedoucí práce Ing. Boženě Mannové, Ph.D., své přítelkyni Bc. Zuzaně Soukupové za podporu a korektury textu a své rodině a přátelům, kteří mi byli a jsou dlouhodobou oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ……. ……

Podpis autora práce

Abstrakt

Tato bakalářská práce se zabývá vytvořením aplikace automatizující plánování, vytváření a obsluhování telefonních hovorů v propojení s VoIP (Voice over IP) serverem.

Abstract

This thesis deals with creating an application that automates planning, creating and controlling telephone calls in connection with a VoIP (Voice over IP) server.

Obsah

1 Úvod

V dnešní době se automatizace telefonních hovorů zabývá převážně automaty, které hovory přijímají a zpracovávají vstup od volajícího. Tuto technologii dnes přijala většina call center větších společností za účelem jednoduchého předání volajícího správnému oddělení bez použití operátorů dedikovaných této činnosti. Tato bakalářská práce se však zabývá uplatněním automatizace telefonních hovorů, které v dnešní době, až na výjimky (viz. kapitola 4.1), není příliš časté, a to automatizací vytváření hovorů a jejich obsluhou od začátku hovoru až do ukončení bez potřeby lidských operátorů.

Náplní této bakalářské práce je návrh a implementace aplikace, která zpracovává dodaná data a dle nich plánuje a vykonává telefonní hovory v dané časy. Obsahem hovoru je sada nahrávek a číselných polí, které se mohou lišit pro jednotlivé obvolávané kontakty. Data hovorů aplikace získává z databáze či přes přímé spojení, ke kterému vystavuje rozhraní. Aplikace hovory vykonává v propojení s VOIP[1] serverem.

Náplní této práce není tvorba aplikace vytvářející data hovorů a/nebo zprostředkovávající uživatelské rozhraní.

[1] Voice Over Internet Protocol – sada technologií a protokolů umožňující přenos hlasu v počítačové síti či přes síť internet

2 Funkční požadavky

„Funkčním požadavkem je formulace toho, co by měl systém dělat – popisuje požadovanou funkci systému.“ (1 str. 80)

- Aplikace v pravidelných intervalech kontroluje databázi a podle záznamů v ní naplánuje hovory
- Nastane-li čas, kdy má hovor proběhnout, aplikace pomocí VOIP serveru vytočí požadované číslo a po zvednutí volaného telefonu přehraje sadu nahrávek specifikovanou v databázi
    - Sada nahrávek může obsahovat i číselná pole, která budou převedena na řeč.
    - Některé nahrávky či číselná pole mohou být různá pro jednotlivé volané kontakty.
- Po ukončení hovoru aplikace zaznamená informace o hovoru do databáze, tzn., zda hovor proběhl (a jeho délku), nebo zda bylo volané číslo obsazené, nedostupné, apod.
- Aplikace vystaví rozhraní pro přijímání jednoduchých příkazů jako například jednorázové okamžité vytvoření hovoru, vypnutí nebo restart aplikace, změna konfigurace, apod.
- Aplikace umožní konfiguraci pomocí textového souboru.

3 Nefunkční požadavky

„Nefunkční požadavek je omezující podmínka uvalená na daný systém.“ (1 str. 80)

- Aplikace musí být napsána v jazyce Java.
- Aplikace musí používat technologii Apache Maven pro řízení závislostí.
- Aplikace musí jako databázi používat MySQL.
- Aplikace musí být pro spuštění na serveru distribuovaná jako JAR[2] balík.
- Jako VoIP server bude aplikace používat open source řešení Asterisk[3].

[2] JAR (Java Archive) je platformně nezávislý formát pro archivaci souborů. (3) Používá se pro distribuci aplikací a knihoven pro JVM (Java Virtual Machine)

[3] Asterisk <http://www.asterisk.org/>

4 Analýza

Tato kapitola se zabývá analýzou problematiky a dostupných technologií.

4.1 Podobné aplikace

Na rozdíl od automatizace příchozích hovorů v současné době ještě není velké množství služeb či produktů, které by poskytovaly možnost hovory automaticky vytvářet a obsluhovat. Zatímco v podstatě každé call centrum používá nějaký software pro automatické vytáčení čísel, jakmile je hovor spojen, je přepojen na operátora. Tato práce se zabývá vytvořením systému, který nepotřebuje lidského operátora.

S jedním příkladem automatizace odchozích hovorů se můžeme setkat např. u společnosti Google[4], která u potvrzení telefonního čísla u účtu kromě standardní možnosti SMS s kódem nabízí také automatický hovor[5], ve kterém je potvrzovací kód přečten automatem. Dalším příkladem je například hovor informující o přistavení objednaného vozidla taxi služby.

Na rozdíl od problému, kterým se zabývá tato práce, však tyto služby využívají pouze jednorázové vytvoření hovoru. Pro automatizované odchozí hovory existuje několik komerčních služeb, která nabízí např. americké společnosti SirsiDynix[6], VoltDelta[7], nebo česká společnost Infocall[8].

[4] <http://google.com>
[5] <https://support.google.com/accounts/answer/114129?hl=en>
[6] <http://www.sirsidynix.com/products/voice-automation>
[7] <http://www.voltdelta.com/voice-self-service-ivr/proactive-outreach/automated-outbound-messaging>
[8] <http://www.infocall.cz/ivr.html>

5 Návrh řešení

Tato kapitola se zabývá návrhem řešení od výběru vhodných technologií po strukturu dat v databázi a zapojení jednotlivých strojů v systému, na kterém bude aplikace nasazena.

5.1 Použité technologie

Tato kapitola obsahuje seznam hlavních použitých technologií spolu se stručným popisem každé z nich. Technologie byly zvoleny na bázi požadavků, předchozích zkušeností a osobních preferencí. Ve většině případů se nejedná o jedinečné technologie a bylo by možné docílit stejných výsledků i s technologiemi jinými.

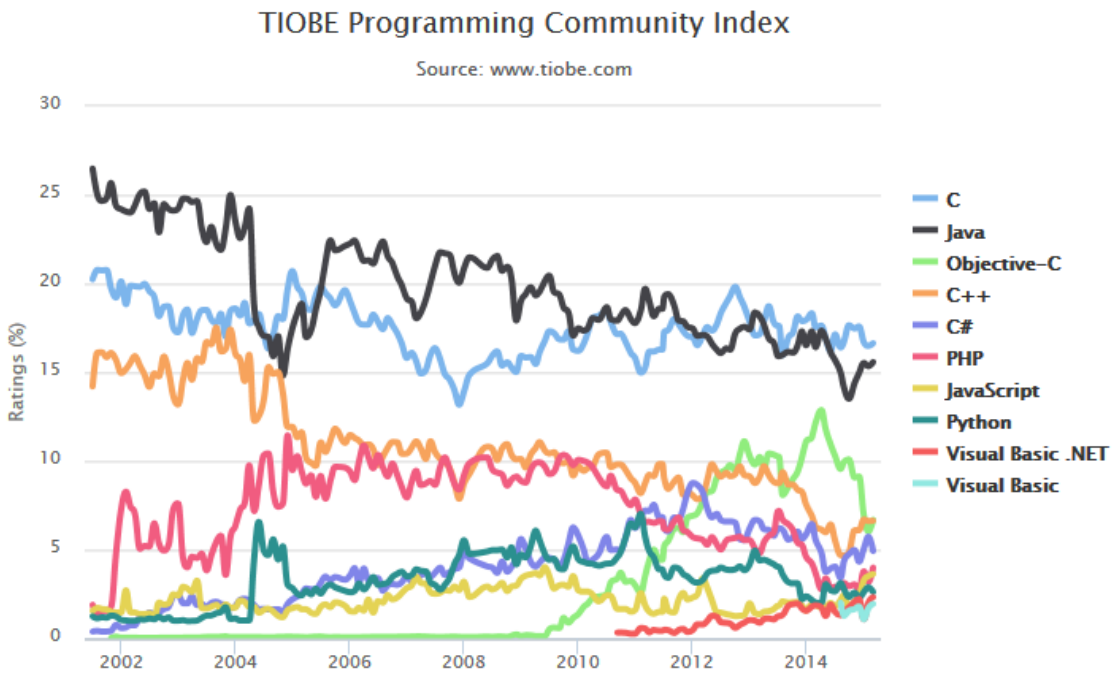
5.1.1 Java

Java je objektově orientovaný programovací jazyk poprvé představen firmou Sun Microsystems v roce 1995[9]. Java je v současnosti jedním z nejpopulárnějších programovacích jazyků, podle TIOBE Indexu je v současné chvíli[10] Java druhým[11] nejpoužívanějším programovacím jazykem s podílem 15,58%, překonána pouze jazykem C s podílem o procento vyšším (celkem 16,64%). Třetí příčku obsazuje Objective-C s podílem téměř o deset procent nižším (celkem 6,69%).

Veliké části své popularity jazyk Java vděčí své platformové nezávislosti. Té je dosáhnuto tím, že zdrojový kód se nekompiluje přímo do strojového kódu, ale do tzv. bajt kódu (bytecode), který jde spustit na jakémkoliv zařízení s interpretem Javy, tzv. JVM[12].

V rámci požadavků této práce nebylo možné zvolit jinou technologii.

[9] What is Java <https://www.java.com/en/download/faq/whatis_java.xml>
[10] Březen 2015
[11] TIOBE <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> vid. 7. 4. 2015
[12] *Java Virtual Machine* - virtuální stroj, který provádí mezikód (bytecode) Javy. Programovací jazyk Java je (nejčastěji) překládán do mezikódu, který provádí právě virtuální stroj. Ten může kód interpretovat, nebo kompilovat pro danou platformu. <http://www.abclinuxu.cz/slovnik/jvm>



Obrázek 1: TIOBE Index programovacích jazyků v letech 2002 až po současnost (zdroj tiobe.com)

5.1.1.1 Apache Maven

Apache Maven je mimo jiné nástroj pro správu závislostí v projektech psaných v jazyce Java[13]. Maven sjednocuje a usnadňuje proces buildování a definuje různé osvědčené postupy (angl. best practices), např. ve struktuře projektů, kde je oddělen kód aplikace od kódu testů. Všechna IDE[14] z nejpopulárnější trojice NetBeans, IntelliJ a Eclipse podporují Maven, i když Eclipse pouze pomocí pluginu.

Správa závislostí pomocí Maven umožňuje sdílet kód aplikace bez nutnosti přibalování často objemných knihoven, což velice usnadňuje např. práci v týmu při použití systému pro správu verzí (angl. version control) jako je GIT[15]. Knihovny jsou definovány v konfiguračním souboru a při spuštění buildu aplikace jsou automaticky stáhnuty z online repozitářů.

Alternativou k této technologii je např. Gradle[16], který nabízí velice podobné možnosti, co se týče správy závislostí. Pro tuto aplikaci je však z důvodu požadavků a předchozích zkušeností zvolena technologie Apache Maven.

[13] Přestože je možné tento nástroj použít i pro projekty v jiných jazycích, podporován je převážně jazyk Java.

[14] *Integrated Development Environment* - česky Vývojové prostředí.

[15] <http://git-scm.com/>

[16] <https://gradle.org/ >

5.1.1.2 Hibernate

Jako ORM[17] vrstva bude použit framework Hibernate[18]. Hibernate umožňuje vytvoření perzistence dat – tedy zachování stavu dat i po ukončení aplikace – pomocí mapování Java objektů na entity v relační databázi. Jedná se o jednu z implementací JPA[19] a usnadňuje komunikaci s databázovými systémy. Mimo jiné poskytuje i dotazování pomocí HQL[20], který vychází z SQL[21].

Jednou z alternativ k Hibernate je Spring JDBCTemplate. Oproti Hibernate je tato knihovna menší, nabízí však tedy i méně funkcionality a tudíž vyžaduje více kódu. Hibernate naopak umožňuje rychle a stručně napsat požadovanou funkcionalitu.

Z důvodů předchozích zkušeností a osobních preferencí bude použita knihovna Hibernate.

5.1.1.3 Apache Log4J

Pro účely logování byla zvolena knihovna Log4j[22]. Knihovna umožňuje logování na několika úrovních a pomocí nastavení logovací úrovně je možné změnit, které zprávy budou a nebudou logovány. Aplikace bude logovat zároveň na standardní výstup a do souboru.

V současné chvíli je již dostupná verze Log4j 2. V době implementace však ještě dostupná nebyla, a tudíž je použita verze 1.2. Vzhledem k absenci zpětné kompatibility verze 2 s předchozími verzemi není upgrade na novou verzi součástí této práce.

Alternativou je například knihovna Logback[23], vytvořena původním autorem knihovny Log4J, a poskytující podobnou funkcionalitu a srovnatelný výkon[24]. Z důvodů předchozích zkušeností byla zvolena technologie Apache Log4J.

5.1.1.4 Google GSON

Google GSON[25] je aplikace sloužící k převodu POJO[26] na JSON a obráceně. Na rozdíl od většiny jiných knihoven sloužících stejným účelům však nepotřebuje v třídách žádné anotace a tudíž lze použít i na třídy, k jejichž zdrojovému kódu není přístup.

---

[17] Objektově-relační mapování (Object-relational mapping)
[18] <http://hibernate.org/>
[19] Java Persistence API <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>
[20] Hibernate Query Language
[21] *Structured Query Language*- česky Strukturovaný dotazovací jazyk
[22] <http://logging.apache.org/log4j/1.2/>
[23] <http://logback.qos.ch/>
[24] <http://logging.apache.org/log4j/2.x/manual/async.html>
[25] <https://code.google.com/p/google-gson/>
[26] Plain Old Java Object – obyčejný Java objekt

Jednou z nejpopulárnějších alternativ je Jackson[27]. Knihovna GSON byla zvolena na základě osobních preferencí po vyzkoušení obou knihoven.

5.1.1.5 Joda Time

Joda Time[28] je knihovna rozšiřující standardní Java třídy pro čas a datum a nabízí spoustu nových tříd a metod pro rychlejší a snadnější práci s nimi.

Alternativou je např. Date4J[29], který také poskytuje rychlejší a snadnější práci s časem a daty. Na rozdíl od Joda Time je Date4J velice minimalistickou knihovnou. Z důvodů předchozích zkušeností a osobních preferencí byla použita knihovna Joda Time.

5.1.1.6 Ari4Java

Knihovna Ari4Java[30] slouží k abstrahování komunikace Java aplikace s Asterisk serverem pomocí ARI[31] a je tvořena částečně z ručně psaného kódu a částečně z kódu automaticky vygenerovaného ze Swagger[32] definic ARI rozhraní. Knihovna je vydána pod licencí GNU LGPL[33].

V době implementace aplikace nebyla dostupná žádná alternativní knihovna abstrahující komunikaci s Asteriskem pomocí ARI pro jazyk Java.

5.1.2 MySQL

MySQL je relační databázový systém založený na jazyku SQL. MySQL využívá model dvojitého licencování a kromě zdarma dostupně open source verze poskytuje i placenou enterprise verzi, ke které mimo jiné nabízí i 24/7 asistenci a dobu reakce do 30 minut.

Podle výzkumu zveřejněného společností Embarcadero na konci roku 2010 tvoří databáze MySQL asi 37,9% trhu[34].

Alternativ k MySQL je veliké množství. Databáze typu MySQL byla zvolena z důvodu požadavku.

---

[27] <http://jackson.codehaus.org/ >
[28] <http://www.joda.org/joda-time/>
[29] <http://www.date4j.net/>
[30] <https://github.com/l3nz/ari4java>
[31] Asterisk REST Interface
[32] <http://swagger.io/>
[33] GNU Lesser General Public License <http://www.lgpl.cz/cesky--preklad-licence--gnu-lesser--general-public-license--v-3-0.php>
[34] Database survey report <http://www.embarcadero.com/images/dm/technical-papers/database-survey-report.pdf>

5.1.3 Asterisk

Asterisk je open source framework pro vytváření komunikačních aplikací, poprvé vydán pod GPL[35] licencí v roce 1999[36] Markem Spencerem a v současné době je vyvíjen a udržován celosvětovou komunitou tisíců vývojářů.

V tomto projektu je použita verze 12 vydaná jako release candidate 20. 12. 2013. Tato verze byla pro vývoj upřednostněna kvůli zavedení komunikační vrstvy ARI[37], která usnadňuje komunikaci aplikací s Asterisk serverem pomocí REST API[38] oproti předchozím AGI[39] a AMI[40]. Zatímco ARI nenahrazuje plně funkcionalitu AGI a AMI, vyhovuje požadavkům tohoto projektu lépe. Novější verze pravděpodobně budou také fungovat, není to však otestováno.

Jednou z alternativ je FreeSWITCH[41], opensource VOIP platforma nabízející funkcionalitu srovnatelnou s Asteriskem. Pro komunikaci s aplikacemi třetích stran jsou dostupné knihovny, a to včetně konektoru pro jazyk Java. Práce s knihovnou Ari4Java je však jednodušší a Asterisk má kolem sebe větší komunitu uživatelů a tudíž více dodatečných knihoven apod. Z tohoto důvodu, a z důvodu požadavků, byla zvolena technologie Asterisk.

5.1.4 GSM Brána

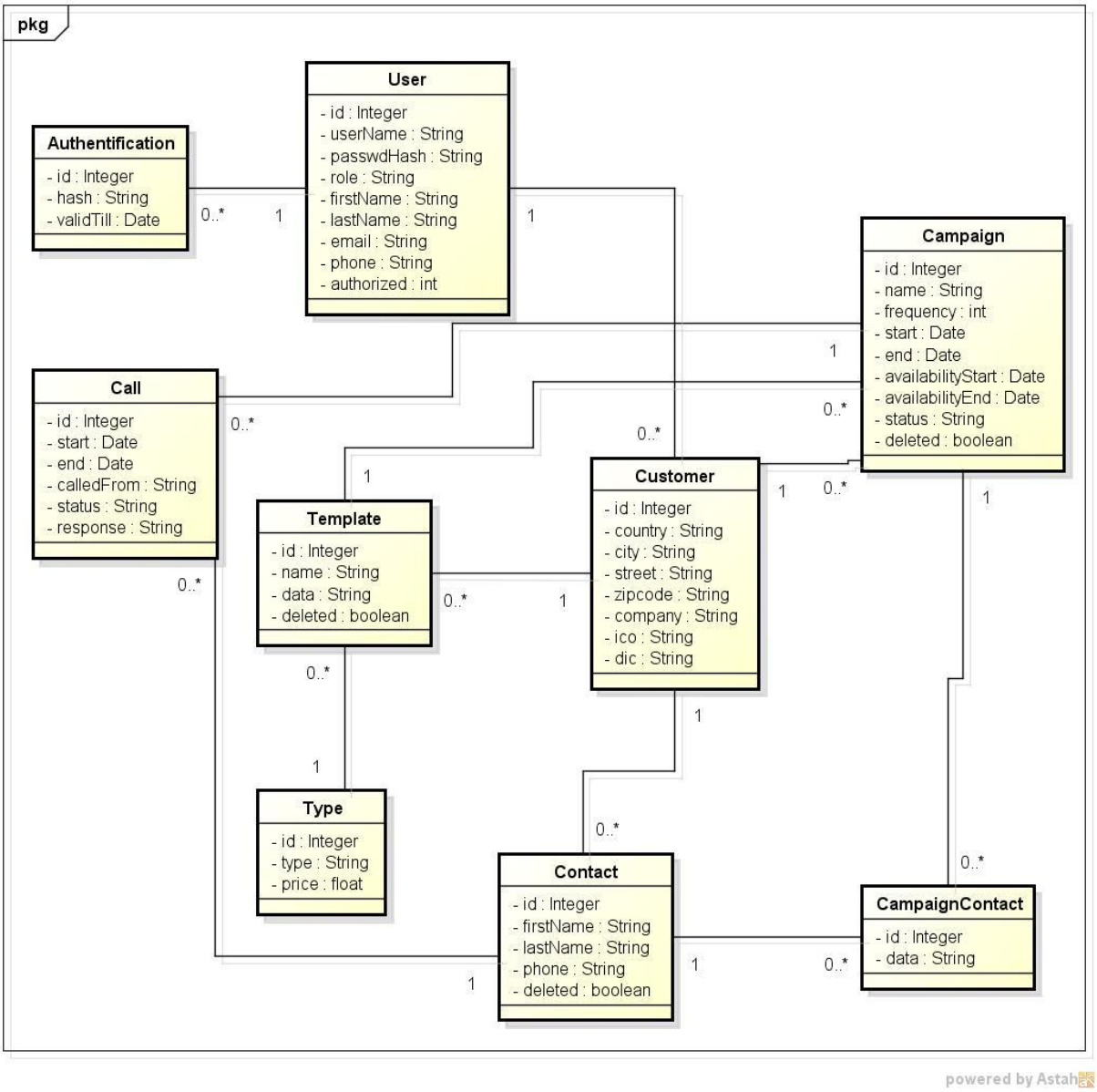
GSM brána (angl. GSM Gateway) je hardwarová komponenta umožňující připojení do mobilní GSM sítě. Za tímto účelem mívají jeden či více otvorů pro vložení standardních SIM karet.

Během vývoje a testování této aplikace byl Asterisk server připojen k GSM bráně StarGate VOIP od společnosti 2N Telecommunications[42] se dvěma SIM kartami. Aplikace nevyžaduje konkrétně tento produkt a tudíž nastavení GSM brány není součástí návodu v příloze 9.2. Možné je použít jakoukoliv technologii umožňující SIP[43] VOIP[44], takže lze využít i např. služeb VOIP operátorů[45].

[35] General Public License <http://www.gnugpl.cz/v3/>
[36] <http://www.asterisk.org/get-started>
[37] Asterisk REST Interface
[38] <http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>
[39] Asterisk Gateway Interface
[40] Asterisk Manager Interface
[41] <https://freeswitch.org/>
[42] <http://www.2n.cz/cz/produkty/gsm-brany/voip-gsm-brany/stargate-voip/>
[43] Session Initiation Protocol - česky protokol pro inicializaci relací <http://www.en.voipforo.com/SIP/SIP_architecture.php>
[44] Voice Over Internet Protocol - <http://www.joyce.cz/co-je-voip/>
[45] Příklad jednoho z mnoha VOIP operátorů v České Republice <http://voip.mikrotech.cz/>

5.2 Doménový model

Doménový model je diagramem tříd představující databázové entity a jejich vzájemné vztahy.



Obrázek 2: UML[46] schéma doménového modelu

5.3 Návrh fyzického systému

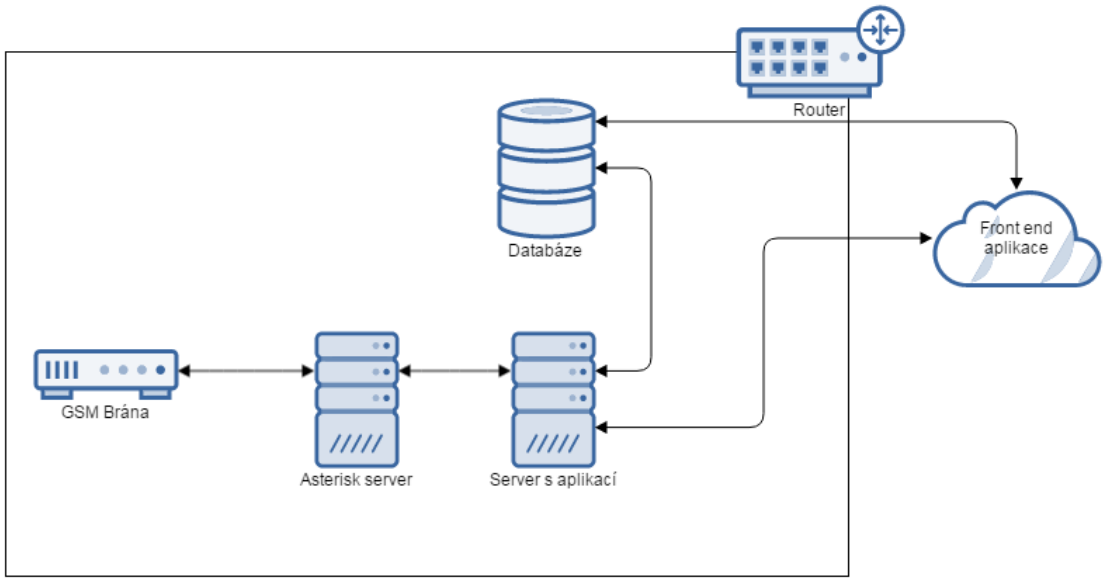
Fyzickým systémem se zde rozumí všechny fyzické stroje sloužící k běhu aplikace a jejich vzájemné propojení.

Pro nasazení by server s Asteriskem, server s aplikací a GSM brána měly být ve stejné logické podsíti. Pokud budou fyzicky v odlišné síti, bude třeba je spojit pomocí VPN[47] nebo přesměrovat všechny příslušné porty a ideálně spojení i zabezpečit.

[46] Unified Modeling Language
[47] Virtual Private Network <https://technet.microsoft.com/cs-cz/library/cc731954(v=ws.10).aspx>

Asterisk server a server s aplikací jsou ve schématu Obrázek 3 navrženy jako dva odlišné stroje, může se však jednat i o jeden jediný, na kterém běží obě aplikace zároveň. Rozdělení však poskytuje některé výhody jako například lepší rozložení výkonu, možnost mít připraven i záložní Asterisk server pro případ výpadku primárního, či použití různých operačních systémů.

Asterisk je v současné chvíli dostupný pouze pro Unixové systémy, zatímco aplikace vyvíjená v rámci této práce je díky využití technologie Java multiplatformní, a lze ji tedy spustit na jakémkoliv systému s Java Virtual Machine.



Obrázek 3: Síťové schéma fyzického systému

6 Implementace

Tato kapitola se zabývá implementací aplikace vycházející z návrhu řešení uvedeného v kapitole 5.

6.1 Struktura projektu

Projekt, resp. Java Maven projekt, je soubor tříd a konfiguračních souborů tvořících aplikaci. Třídy jsou v projektu rozděleny do balíků, tzv. packages, kde každý balík obsahuje třídy tvořící logický celek podle funkcí, které vykonávají.

6.1.1 Balík cz.improvisio.autodialer

Hlavní balík projektu. Poskytuje primární funkcionalitu a obsahuje všechny ostatní balíky. Obsahuje následující třídy:

- AppSingleChecker.java
  - Třída, která při spuštění aplikace kontroluje, zda jiná instance aplikace již neběží na daném stroji.
- Autodialer.java
  - Hlavní třída aplikace. Obsahuje vstupní bod aplikace, tedy metodu *main*.
  - Založeno na návrhovém vzoru singleton [5].
  - Obsahuje metody řešící funkcionalitu zastavování a spouštění dalších komponent, zpracovávání kampaní volání z databáze, vkládání naplánovaných hovorů do fronty a vyjímaní hovorů z fronty ve správný čas.
- DbFetcher.java
  - Třída, jejíž instance běží v samostatném vlákně a v pravidelných intervalech kontroluje stav databáze pro přidané či změněné kampaně.
- ExceptionReporter.java
  - Třída obsahující statickou metodu volanou v případě vyhození kritické výjimky. Asynchronně odešle e-mailovou zprávu s detaily výjimky.
- JData.java a JRecord.java
  - POJO třídy pro deserializaci JSON[48] objektů obsažených v databázi.
- MyAri.java
  - Třída, která pomocí knihovny Ari4Java vybuduje připojení k Asterisk serveru pomocí ARI.
  - Obsahuje metodu pro vytváření hovorů.

[48] JavaScript Object Notation <http://www.json.org/json-cz.html>

- ScheduledCall.java
    - Třída reprezentující naplánovaný hovor.
    - Založeno na návrhovém vzoru factory [5], který je použit, aby každý naplánovaný hovor měl jedinečný identifikátor a byl tímto poté jednoduše dohledatelný při přijímání odpovědí od Asterisku.
- StasisAD.java
    - Třída reprezentující hovor ve stavu „stasis“, tedy hovor, který byl přijat.
    - Obstarává správné přehrávání nahrávek během hovoru.
- States.java
    - Enum[49] reprezentující možné stavy aplikace. Příkladem možných stavů jsou např. RUNNING, PAUSED, RESTARTING, apod.
- Utils.java
    - Třída obsahující několik užitečných statických metod jako například převod milisekund na dny, převod typu nahrávky do tvaru použitelným Asteriskem, či odstranění diakritiky z řetězce, který má být odeslán SMS zprávou.
- WsClientHandler.java
    - Třída zpracovávající příchozí websocketovou komunikaci od Asterisk serveru, jako např. začátek či konec stáze hovoru, dokončení přehrávání dané nahrávky, apod.

6.1.2 Balík cz.improvisio.autodialer.cmdserver

Balík obsahující třídy patřící ke komponentě umožňující aplikaci přijímat příkazy pomocí komunikace přes sockety. Obsahuje následující třídy:

- ClientConnection.java
    - Třída reprezentující přijaté připojení klienta.
    - Přijme a zpracuje zprávu ve vlastním vlákně a odešle odpověď.
- CmdServer.java
    - Třída reprezentující server naslouchající na portu pro příchozí komunikaci.
    - Běží ve vlastním vlákně.
- Credentials.java
    - Třída implementující funkcionalitu pro ověření autentizačních údajů odesílatele příkazu.

[49] Výčtový typ <http://www.algoritmy.net/article/30320/Enum-19>

- JMessage.java
    - Třída implementující funkcionalitu pro deserializaci přijatého příkazu ve formátu JSON.
- ResponseMessage.java
    - Třída implementující funkcionalitu pro vygenerování odpovědi serveru ve formátu JSON.

6.1.3 Balík cz.improvisio.autodialer.cmdserver.Message

Balík obsahující třídy reprezentující jednotlivé typy přijaté zprávy a rozhraní (interface) Message, které všechny třídy zpráv implementují.

6.1.4 Balík cz.improvisio.autodialer.configuration

Balík obsahující třídy používané pro ukládání a načítání konfigurace. Obsahuje následující třídy:

- Configuration.java
    - Během běhu aplikace obsahuje veškeré konfigurační hodnoty.
    - Obsahuje metody pro změnu a získání hodnot jednotlivých konfiguračních položek.
    - Umožňuje konfiguraci načítat z a ukládat do XML[50] souboru.
- Options.java
    - Obsahuje pole řetězců obsahující názvy všech konfigurovatelných položek.

6.1.5 Balík cz.improvisio.autodialer.db

Balík obsahuje třídu HibernateUtil.java vygenerovanou knihovnou Hibernate.

6.1.6 Balík cz.improvisio.autodialer.db.entities

Balík obsahuje třídy reprezentující databázové entity mapované pomocí ORM.

6.1.7 Balík cz.improvisio.autodialer.db.entities.dummies

Obsahuje několik jednoduchých dummy tříd – tedy tříd určené pro použití v zastoupení jiných – nutných pro hovory vytvářené z příkazů, tedy hovorů, které nepatří k žádné kampani, žádnému zákazníkovi, ani nejsou vytvářeny pro žádný kontakt existující v databázi.

Tyto třídy implementují metody entit, které jsou volány v průběhu životního cyklu naplánovaného hovoru.

[50] E**X**tensible **M**arkup **L**anguage <http://www.w3schools.com/xml/xml_whatis.asp>

6.2 Životní cyklus aplikace

Tato kapitola popisuje chování aplikace od jejího spuštění do jejího ukončení včetně všech vedlejších procesů či vyvolaných aktivit, jako například přijetí příkazu.

6.2.1 Inicializace

V prvním kroku inicializační fáze aplikace zkontroluje, zda již na stejném stroji neběží jiná instance aplikace pomocí komponenty AppSingleChecker. Tohoto je docíleno otevřením socketu na portu 9999, který byl vybrán kvůli neobvyklosti jeho užití[51] v jiných aplikacích. Jestliže otevření selže, pravděpodobně již běží jiná instance aplikace.

Dále je aplikace přepnuta do stavu *STARTING*, jestliže je inicializována nově (tzn. nové spuštění aplikace) či ze stavu *PAUSED*. V případě, že je aplikace restartována, nachází se stále ve stavu *RESTARTING*.

V druhém kroku inicializace aplikace načte hodnoty nastavení z konfiguračního souboru, případně jsou nastaveny na výchozí hodnotu, chybí-li. Nastaví se úroveň logování.

Ve třetím kroku je inicializována komponenta CmdServer, sloužící pro přijímání příkazů. Příkazový server ve svém vlastním vlákně otevře socket pro naslouchání na konfigurací daném portu.

V kroku čtvrtém je inicializována samotná instance hlavní třídy. Instance naváže spojení s databází a následně s Asterisk serverem. Kvůli kritičnosti správného připojení k Asterisku je v případě selhání navázání spojení pokus opakován v pětisekundových intervalech, dokud se spojení nepovede navázat.

V pátém kroku probíhá načítání kampaní z databáze a naplánování hovorů (viz. 4.2.2)

V šestém kroku je inicializována komponenta DbFetcher, která ve vlastním vlákně kontroluje v pětiminutových intervalech stav databáze pro změny v kampaních.

Nakonec je aplikace přepnuta do stavu *RUNNING* a přechází do hlavního cyklu aplikace (viz. kapitola 6.2.3)

6.2.2 Načítání kampaní z databáze a plánování hovorů

V prvním kroku jsou z databáze načteny všechny kampaně, které v současné chvíli běží a nejsou smazané. Pokud jsou již naplánované hovory pro nějaké kampaně, tyto hovory se odstraní z fronty.

V druhém kroku se iteruje přes všechny kampaně z kroku prvního. Pro každou kampaň je nejdříve zjištěn čas, kdy se má vykonat další hovor. Je-li perioda volání záporná či

[51] <http://www.speedguide.net/port.php?port=9999>

rovna nule, jedná se pouze o jednorázový hovor, a tudíž se čas hovoru naplánuje na současný čas. V případě, že kampaň ještě nezačala, naplánuje se hovor na čas uvedený jako začátek kampaně.

V případě, že kampaň již běží a má kladnou frekvenci, zjistí se nejprve, zda v poslední periodě již byly provedeny nějaké hovory. Pokud ještě žádné provedeny nebyly, naplánují se všechny hovory této kampaně na současný čas. V opačném případě budou již provedené hovory naplánovány až na další periodu a hovory ještě neuskutečněné na současný čas.

Dále se kontroluje dostupnost naplánovaného času. Kampaň má čas od kdy do kdy se smí volat a tudíž jestli byl nějaký hovor naplánován mimo tuto dobu, je přeplánován na začátek dostupnosti kampaně.

Pokud je v tomto kroku zjištěno, že byl hovor naplánován na čas po konci kampaně, je kampaň uzavřena.

Ve třetím kroku plánování je pro každý hovor naplánovaný v předchozích krocích z databáze načtena šablona hovoru a jednotlivé položky šablony jsou převedeny do formátu srozumitelného Asterisk serveru a přidány do fronty nahrávek naplánovaného hovoru. Pokud šablona obsahuje proměnlivé položky závislé na konkrétním kontaktu, jsou v tomto kroku doplněny správné hodnoty.

V posledním kroku jsou naplánované hovory přidány do fronty. Fronta naplánovaných hovorů je implementována pomocí třídy PriorityQueue[52], kde jednotlivé hovory jsou porovnávány podle naplánovaného času jejich uskutečnění.

6.2.3 Hlavní cyklus aplikace

Hlavní cyklus aplikace běží, dokud se aplikace nachází ve stavu *RUNNING*. Ve vteřinových intervalech je kontrolován stav fronty hovorů a čeká se, dokud nenastane čas dalšího naplánovaného hovoru. Jakmile tento čas nastane, je naplánovaný hovor vyjmut z fronty a provede se hovor.

6.2.4 Průběh hovoru

V prvním kroku je hovor vyjmut z fronty a je zjištěno, zda od naplánování hovoru neproběhlo k ukončení či pozastavení kampaně.

V druhém kroku proběhne kontrola, zda již neprobíhá jiný hovor na stejné číslo (např. obsahuje-li více kampaní stejný kontakt). Pokud ano, je hovor odložen o 10 vteřin.

[52] <https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html>

Ve třetím kroku proběhne kontrola, zda zákazník, kterému patří kampaň spojená s tímto hovorem, má dostatek kreditu na uskutečnění tohoto hovoru. Pokud ne, je kampaň pozastavena s příznakem *no-credit*.

Ve čtvrtém kroku je hovor vytočen a přidán do seznamu aktivních hovorů. Pokud je vytvořením tohoto hovoru dosáhnut maximální povolený počet souběžně probíhajících hovorů, vyčká se na ukončení některého z probíhajících hovorů.

Pátý krok nastává, je-li hovor spojen (tzn., volaný zvedl telefon). Pokud se hovor nepovede spojit, přejde se rovnou k sedmému kroku. V chvíli, kdy je hovor úspěšně spojen, je inicializována tzv. stáze hovoru, která přiřazuje zprávy od Asterisk serveru ke správným probíhajícím hovorům.

V šestém kroku alternuje odeslání požadavku o přehrání nahrávky s přijímáním zpráv o dokončení přehrávání. Toto pokračuje, dokud se nedokončí přehrávání poslední nahrávky, či do přijetí zprávy o ukončení hovoru volaným. Po ukončení hovoru je odečten příslušný kredit.

V sedmém kroku se do databáze zanesou údaje o hovoru – čas začátku, čas konce, zda byl úspěšně spojen, atd.

V posledním kroku je hovor znovu naplánován na příští periodu.

6.2.5 Přijímání příkazů

Příjem a zpracování příkazu začíná přijetím připojení na portu příkazového serveru.

V prvním kroku je serverem vytvořeno nové vlákno spojení, ve kterém bude celá komunikace obsloužena.

V druhém kroku jsou po řádcích čtena data zprávy.

Ve třetím kroku jsou zkontrolovány autentifikační údaje ve zprávě. Autentifikace probíhá pomocí databáze. Klient, zasílající příkazovou zprávu, do databáze uloží uživatele, hash[53] a datum, kdy platnost tohoto skončí. Server tedy autentizuje porovnáním údajů o uživateli a hashi ve zprávě s údaji v databázi, a zda jsou ještě platné. V případě, že údaje nejsou správné, přejde se k šestému kroku.

Ve čtvrtém kroku je zpráva deserializována z řetězce obsahující JSON definici objektu na Java třídu reprezentující daný typ zprávy.

V pátém kroku se provede příkaz ve zprávě. Podle typu příkazu se může přejít na šestý krok i před dokončením provedení příkazu a aplikace může přejít do stavu *EXECUTING*.

[53] <http://www.abclinuxu.cz/slovnik/hash>

V šestém kroku je odeslána odpověď obsahující kód odpovědi (využity jsou standardizované HTTP kódy, viz. Příloha 9.3) a případná data, vyžadoval-li to daný příkaz. Nakonec je spojení ukončeno. Některé příkazy vyžadují restart aplikace, a tudíž aplikace může přejít do stavu *RESTARTING*.

6.2.6 Pozastavení, znovuspuštění a restart aplikace

Po přijetí příkazu na pozastavení aplikace je aplikace převedena do stavu *PAUSED* a jsou ukončeny instance hlavní třídy a instance komponenty DbFetcher. Příkazový server ve stavu pozastavení nadále funguje.

Po přijetí příkazu na znovuspuštění aplikace je spuštěna inicializace (viz.kapitola 6.2.1), vyjma prvního kroku.

Po přijetí příkazu na restart aplikace jsou ukončeny všechny běžící instance a spuštěna inicializace (viz. kapitola 6.2.1), vyjma prvního kroku.

6.2.7 Ukončení aplikace

Ke korektnímu ukončení aplikace dojde pouze po přijetí příkazu s požadavkem o ukončení.

7 Testování

Testování funkcionality aplikace probíhalo v několika fázích:

7.1 Unit testy

Unit testy, česky jednotkové testy, je jeden ze způsobů tzv. black-box testování, tedy testování jednotlivých částí aplikace bez znalosti jejich vnitřního chování. Částem aplikace jsou zadána data a kontroluje se správnost dat na výstupu.

Unit testy proběhly na všech podstatných třídách aplikace. Pro implementaci unit testů byl zvolen Framework JUnit.[54] Pro metody vyžadující data z databáze byla do databáze přidána testovací data. Tento postup byl zvolen pro jednoduchost návrhu a implementace, a protože byla možnost samostatné databáze pouze pro testování. Jinou možností by bylo použití tzv. mocku databáze, např. použitím frameworku Mockito.[55]

7.2 Testovací provoz

Aplikace byla během vývoje několikrát nasazena do testovacího provozu na vývojářský server s několika aktivními kampaněmi po dobu jednoho týdne.

Během prvního testovacího provozu vyvstal na povrch problém s vyprcháním (angl. timeout) připojení s databází. Tento problém vychází z maximální doby připojení k databázi typu MySQL a byl vyřešen použitím knihovny C3P0[56].

Při posledních dvou nasazeních do testovacího provozu se nevyskytly žádné neočekávané problémy.

[54] <http://junit.org/>
[55] <http://mockito.org/>
[56] <http://www.mchange.com/projects/c3p0/>

8 Závěr

Výsledná aplikace je funkční a splňuje původní zadání v plném rozsahu. Nad rozsah byla implementována i funkcionalita zasílání jednorázových SMS zpráv z příkazu.

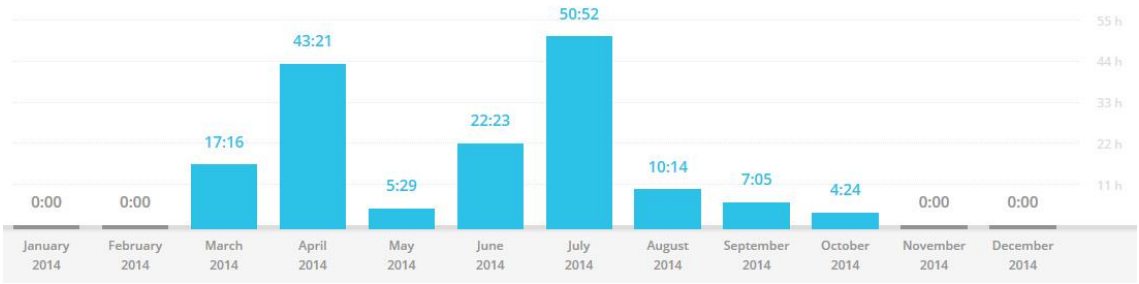
Praktickým využitím této aplikace může být např. pravidelné upomínání zákazníka na neuhrazené faktury, nabídka služeb, obchodní sdělení, apod.

Možné budoucí rozšíření aplikace by mohla být podpora e-mailových a SMS kampaní. Dalším rozšířením by mohla být funkcionalita pro převod textu na mluvenou řeč. Takovouto funkcionalitu nabízí i Asterisk ve formě dodatečných knihoven, v současné době však není žádná takováto knihovna dostupná pro češtinu. Řešením by zde tedy mohl být převod textu do mluvené řeči další komponentou, která by výsledné audio uložila do souboru na serveru a s tím by se pak zacházelo jako s jakoukoliv jinou nahrávkou. Toto by šlo řešit na úrovni této aplikace, či na úrovni klientské aplikace.

9 Přílohy

9.1 Časové výkazy

Na grafu Obrázek 4 lze vidět práce na aplikaci rozdělena do měsíců během vývoje. Časové údaje jsou uvedeny v hodinách. Časové výkazy byly zadávány do online systému Toggl[57], ze kterého je i vygenerován tento graf.



Obrázek 4: Přehled času stráveného prací na této aplikaci

9.2 Instalační návod a konfigurace

Tato kapitola popisuje postup na instalaci a konfiguraci aplikačního a Asterisk serveru tak, aby bylo možné aplikaci používat. Tento návod uvažuje zapojení popsané ve schématu Obrázek 3 a Unixový operační systém jak na aplikačním, tak i na Asterisk serveru.

9.2.1 Asterisk server

Po instalaci Asterisku na server je třeba nastavit ARI, aby se aplikace mohla k serveru připojit. Toto se nastavuje v konfiguračním souboru standardně umístěném v */etc/asterisk/ari.conf*. Vzorové nastavení lze nalézt v příloze Příloha 1. U vzorového nastavení jsou položky umístěné v „<>“ potřeba nahradit vlastními údaji.

```
[general]
enabled = yes
[<uzivatelske jmeno>]
type=user
password=<heslo>
```

Příloha 1: Vzorový obsah souboru /etc/asterisk/ari.conf

Připojení Asterisk serveru ke GSM bráně, případně k jinému poskytovateli telefonie, je třeba nastavit v konfiguračním souboru běžně umístěném v */etc/asterisk/sip.conf*. Vzorové nastavení lze nalézt v příloze Příloha 2.

[57] <https://toggl.com/>

```
[general]
disallow=all
allow=ulaw
allow=alaw
allow=gsm
qualify=yes
canreinvite=no


[sip-phone]
type=friend
context=outgoing
host=<adresa brany>
port=<port brany>
username=<uzivatelske jmeno>
secret=<heslo >
dtmfmode=rfc2833
;nat=force_rport,comedia
insecure=invite,port
allowguest=yes
```

Příloha 2: Vzorový obsah souboru /etc/asterisk/sip.conf

Toto nastavení stačí pro použití aplikace se čtením čísel v angličtině. Chceme-li podporu čtení čísel i česky, je třeba přidat několik řádků nastavení do souboru umístěného v */etc/asterisk/say.conf* (viz. Příloha 3) a umístit potřebné zvukové soubory obsahující čtené číslice ve formátu .gsm do složky */var/lib/asterisk/sounds/cs/digits/*.

```
[cs](digit-base,date-base)
    _[n]um:0. => num:${SAY:1}
    _[n]um:X => digits/${SAY}
    _[n]um:1X => digits/${SAY}
    _[n]um:[2-9]0 =>  digits/${SAY}
    _[n]um:[2-9][1-9] =>  digits/${SAY:0:1}0, num:${SAY:1}
    _[n]um:X00 => digits/${SAY}
    _[n]um:XXX => num:${SAY:0:1}00, num:${SAY:1}
    _[n]um:X000 => digits/${SAY}
    _[n]um:XXXX => num:${SAY:0:1}000, num:${SAY:1}
    _[n]um:XX000 => num:${SAY:0:2}, digits/1000
    _[n]um:XXXXX => num:${SAY:0:2}, digits/1000, num:${SAY:2}
```

Příloha 3: Nastavení potřebné pro české čtení čísel v souboru /etc/asterisk/say.conf

9.2.2 Java aplikace

Java aplikace se zde myslí aplikace, která byla implementována v rámci této bakalářské práce. Zatímco vše ostatní lze nastavit v konfiguračním souboru (viz. Příloha 4), připojení k databázi lze nastavit pouze přímo v projektu, konkrétně v souboru *hibernate.cfg.xml*.

Konfigurační soubor se musí jmenovat *autodialer.cfg.xml* a při nasazení musí být ve stejné složce jako .jar soubor s aplikací.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
    <log-level>ALL</log-level> <!-- available: ALL, TRACE, DEBUG,
INFO, WARN, ERROR, FATAL, OFF-->
    <ari>
        <url>ws://10.0.0.222:8088/</url>
        <user>uzivatel</user>
        <pass>heslo</pass>
        <app-name>Autodialer</app-name>
    </ari>
    <port>4242</port>
    <max-calls>2</max-calls>
    <gsm-url>10.0.0.69</gsm-url>
    <mail>
        <smtp-user>a@example.com</smtp-user>
        <smtp-pass>hesloH35l0</smtp-pass>
        <smtp-host>smtp.example.com</smtp-host>
        <smtp-port>587</smtp-port>
        <smtp-from>info@ example.com</smtp-from>
        <smtp-auth>true</smtp-auth>
        <smtp-tsl>true</smtp-tsl>
        <smtp-set-from> example.com &lt;info@
example.com&gt;</smtp-set-from>
        <smtp-recipient>chyby@example.com</smtp-recipient>
        <smtp-reply-to>info@ example.com</smtp-reply-to>
    </mail>
</config>
```

Příloha 4: Vzorový obsah konfiguračního souboru aplikace

Následuje popis všech polí konfigurace aplikace:

- "log-level" - úroveň logování (může být ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF
- "url" – adresa, na které se nachází Asterisk server (i s protokolem a portem, např. ws://10.0.0.222:8088)
- "user" - uživatel pro přihlášení k ARI
- "pass" - heslo k uživateli pro přihlášení k ARI
    - Uživatelské jméno a heslo pro ARI jsou definovány na Asterisk serveru v souboru */etc/asterisk/ari.conf*
- "app-name" - jméno, pod kterým se aplikace zaregistruje na Asterisk serveru (výchozí hodnota *„Autodialer“*)
- "port" - port, na kterém poběží CmdServer, který přijímá příkazy zaslané aplikaci
- "max-calls" - maximální počet paralelně probíhajících hovorů. Výchozí hodnota je *2*. Toto číslo musí být stejné nebo menší než počet SIM karet v bráně
- "gsm-url" - adresa k GSM bráně (výchozí hodnota 10.0.0.69)
- "smtp-user" - uživatelské jméno k SMTP serveru
- "smtp-pass" - heslo k SMTP serveru
- "smtp-host" - adresa SMTP serveru
- "smtp-port" - port SMTP serveru
- "smtp-from" - položka "from" v emailu
- "smtp-auth" - zda se má používat autentifikace při přihlášení k SMTP serveru (hodnota *true/false*)

- "smtp-tsl" - zda se má používat TSL při přihlášení k SMTP serveru (hodnota *true/false*)
- "smtp-set-from" - jméno odesílatele emailu ve formátu "Moje Jmeno <info@example.com>"
    - Protože konfigurační soubor je XML dokument, je třeba nahradit „<“ za „<“ a „>“ za „>“
- "smtp-recipient" - cílová adresa emailů obsahujících chybová hlášení
- "smtp-reply-to" - reply-to položka emailu

9.3 Formát příkazů pro aplikaci

Způsob, jakým aplikace přijímá a zpracovává příkazy, je popsán v kapitole 6.2.5. Tato kapitola se zabývá konkrétní definicí struktury požadavků a popisem jednotlivých dostupných příkazů.

Požadavky jsou zasílány ve formátu JSON. Při každém připojení je možné zaslat pouze jeden požadavek. Formát požadavku je popsán v příloze Příloha 5.

```
{
  "user":<uzivatel>,
  "hash":<hash>,
  "cmd":<jméno příkazu>,
    <argumenty příkazu>
}
```

Příloha 5: Obecný formát zprávy požadavku

Popis jednotlivých příkazů:

- "update"
    - Znovu načte data o konkrétní kampani.
    - Po přijetí příkazu nejdříve z fronty hovorů odstraní všechny naplánované hovory náležící této kampani (aby se vyhnulo případným duplicitám), následně znovu načte kampaň a naplánuje ji hovory nové.
    - Argumenty:
        - "id":<id kampaně k načtení>
- "call"
    - Vytočí co nejdříve číslo a přehraje sadu nahrávek.
    - Argumenty:
        - "number":<číslo ve tvaru 00xxxxxxxxxxx> (například.: 00420777123456)
        - "records":[{"id":<název>,"type":<typ>,"value":<url nahrávky nebo číslo či číslice k přečtení>},...]
            - Typy jsou: *„record“, „number“* a *„digits“*.
            - Url by měla být absolutní cesta k nahrávce a každé lomítko předchází zpětné lomítko.
- "sms"
    - Pošle SMS na dané číslo obsahující text specifikovaný v příkazu.
    - Argumenty:

        - "number":<číslo ve tvaru 00xxxxxxxxxxx> (například: 00420777123456)
        - "text": text SMS zprávy
- "shutdown"
    - Vypne celou aplikaci, následné lze aplikaci znovu spustit pouze ručně přímo na serveru.
    - Argumenty:
        - žádné
- "restart"
    - Restartuje aplikaci.
    - Argumenty:
        - žádné
- "conf"
    - Změní nastavení (pro provedení všech změn kromě změny úrovně logování je potřeba aplikaci restartovat - toto se provádí automaticky, tj. není nutné následně posílat příkaz k restartu).
    - Argumenty:
        - "entries" - pole objektů obsahující:
            - "var"
                - Jméno položky, která bude měněna.
                - Hodnoty jsou shodné s hodnotami konfiguračního souboru popsaných v kapitole 9.2.2.
            - "val"
                - Hodnota, na kterou se nastaví položka specifikovaná v poli *„var“*.
- "get-conf"
    - Vrátí současnou hodnotu nějakého nastavení.
    - Argumenty:
        - "var"
            - Jméno položky, jejíž hodnota se požaduje.
            - Hodnoty jsou shodné s hodnotami konfiguračního souboru popsaných v kapitole 9.2.2.
- "pause"
    - Přesune aplikaci do stavu *PAUSED*, viz. kapitola 6.2.6.
    - Argumenty:
        - žádné
- "unpause"
    - Znovu spustí aplikaci nacházející se ve stavu *PAUSED*, viz. kapitola 6.2.6.
    - argumenty
        - žádné
- "status"
    - Vrátí současný stav, ve kterém se aplikace nachází.
    - Může nabývat následujících hodnot:
        - RUNNING - běží
        - PAUSED – aplikace je pozastavena
        - SHUTTING_DOWN - aplikace se vypíná
        - STARTING - aplikace se zapíná
        - RESTARTING - aplikace se restartuje

- pozn.: u *STARTING*, *RESTARTING* a *SHUTTING_DOWN* v nějakých chvílích neběží ani CmdServer
        - EXECUTING - aplikace právě provádí nějaký příkaz
    - Argumenty:
        - žádné

Na každý požadavek je odeslána odpověď nesoucí kód odpovědi a případná požadovaná data. Formát odpovědi je popsán v příloze Příloha 6.

```
{"code":<kód odpovědi>,<požadovaná data>}
```

Příloha 6: Všeobecný formát zprávy odpovědi

Pro kód odpovědi se používají standardní HTTP kódy. V praxi se však odesílají pouze následující:

- 200 OK – vše je v pořádku, posílám požadovaná data
- 202 ACCEPTED – vše je v pořádku, příkaz se provede
- 400 BAD REQUEST – chybná syntaxe zprávy
- 404 NOT FOUND – např. když je příkazem *„get-conf“* požadována hodnota nastavení, které neexistuje
- 403 FORBIDDEN – chybí přihlašovací údaje nebo jsou chybné

9.4 Formát dat v databázi

Tato kapitola popisuje syntaxi dat v databázi ve formátu JSON. Takováto data jsou ukládána v entitách **Template** a **CampaignContact** ve sloupci *„data“*.

Formát šablony nahrávek v **Template** obsahuje posloupnost jednotlivých nahrávek a číselných polí. Ve formátu JSON se jedná o pole objektů. Každý takovýto objekt reprezentuje jednu nahrávku či číselné pole a má následující formát:

- type
    - Povinné pole.
    - Může nabývat následujících hodnot:
        - digits - číslo čtené po číslicích
        - number – číslo čtené jako číslo
        - record - nahraný zvukový soubor (část nahrávky)
- value
    - Pole povinné pouze u typu *„record“*.
    - Jedná se o hodnotu objektu.
        - V případě nahrávky se jedná o url k nahrávce.
        - V případě digits nebo number se jedná o číslo, které má být čteno.

- Jestliže objekt neobsahuje toto pole, hodnota se přiřazuje pomocí id a páruje se ke konkrétnímu kontaktu (tzn. každý kontakt zde má jiný obsah).

- id
  - Povinné pole.
  - Označuje objekt.
  - V případě, že objekt neobsahuje pole *„value“*, páruje se pomocí id hodnota pro daný kontakt.
- label
  - Nepovinné pole.
  - Může být použito např. pro označení objektu ve front-end aplikaci.

```
[
{"id":"template1","type":"record","value":"\/var\/lib\/asterisk
\/sounds\/cs\/hlasky\/dobry_den"},
{"id":"template2","type":"digits","value":"124"},
{"id":"template3","type":"digits","label":"\u010d\u00edsl
faktury"},
{"id":"template4","type":"record","value":"\/var\/lib\/asterisk
\/sounds\/cs\/financni_informace\/dekujeme"}
]
```

Příloha 7: Vzorová data šablony

Data v entitě **CampaignContact** přiřazují proměnlivá pole šablony jednotlivým kontaktům a nastavují jim hodnotu. Jedná se o pole objektů, kde každý obsahuje následující položky:

- id
  - Id objektu, musí být stejné jako id objektu v šabloně, do kterého se doplňuje hodnota.
- value
  - Hodnota, která má být doplněna.

```
[{"id":"template2","value":"445566"}]
```

Příloha 8: Vzorová data párování dat šablony s kontaktem

10 Zdroje

1. **Neustadt, Ila a Arlow, Jim.** *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* Brno : Computer Press, 2011. ISBN 978-80-251-1503-9.

2. **Novotný, Luděk.** Historie a vývoj jazyka Java. [Online] 30. 11 2003. [Citace: 7. 4 2015.] http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xnovotn8.htm.

3. **Binary Runner.** Java na Linuxu, Úvod. *Root.cz.* [Online] 23. 5 2002. [Citace: 7. 4 2015.] http://www.root.cz/clanky/java-na-linuxu.

4. **Jordan, Matt.** Asterisk 12 Documentation. [Online] Digium, Inc, 20. 12 2013. [Citace: 7. 4 2015.] https://wiki.asterisk.org/wiki/display/AST/Asterisk+12+Documentation.

5. **Pecinovský, Rudolf.** *Návrhové vzory - 33 vzorových postupů pro objektové programování.* místo neznámé : Computer Press, 2007. ISBN 978-80-251-1582-4.

6. **Pressman, Roger a Maxim, Bruce.** *Software Engineering: A Practitioner's Approach.* místo neznámé : McGraw-Hill, 2014. ISBN-13: 978-0078022128 ISBN-10: 0078022126.

7. **Sonatype Company.** *Maven: The Definitive Guide.* místo neznámé : O'Reilly, 2008. ISBN-13: 978-0596517335 ISBN-10: 0596517335.

Seznam obrázků