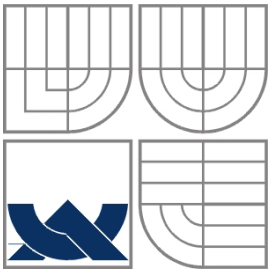
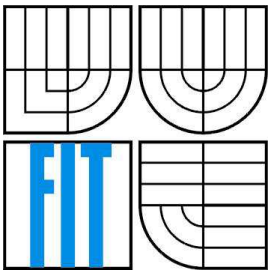


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KLIENT PRO JABBER VYUŽÍVAJÍCÍ KONTEXTOVÉ INFORMACE

CONTEXT-AWARE JABBER CLIENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ GRYC
AUTHOR

VEDOUCÍ PRÁCE — Ing. RUDOLF KAJAN
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá vývojem aplikace sloužící jako klient pro XMPP/Jabber využívající kontextové informace pro mobilní platformu Android. Aplikace umožňuje uživateli prostřednictvím uživatelského rozhraní přijímat a odesílat zprávy, zobrazit seznam kontaktů a podporuje více souběžných konverzací.

Abstract

This thesis is focused on development of application used as context-aware XMPP/Jabber client for Android platform. The application allows the user to send and receive messages, show the contacts list via the user´s interface, it also supports multiple simultaneous conversations.

Klíčová slova

XMPP, Jabber, kontext, instant messenger, klient

Keywords

XMPP, Jabber, context, instant messenger, client

Citace

Gryc Lukáš: Klient pro Jabber využívající kontextové informace, bakalářská práce, Brno, FIT VUT v Brně, 2013

Klient pro Jabber využívající kontextové informace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Kajana.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Chtěl bych poděkovat panu Ing. Rudolfu Kajanovi za cenné rady týkající se zejména výběru technologií použitých k implementaci aplikace.

Obsah

1 Úvod

V posledních letech zažil svět mobilních telefonů a mobilního internetu velký rozmach. Datový tarif má v dnešní době aktivní velká část vlastníků „chytrých“ mobilních telefonů a bezdrátové připojení je dostupné na mnoha veřejných místech. Není tedy divu, že lidé začali používat rychlejší a hlavně levnější způsob komunikace, než kterým jsou SMS – instant messaging[1].

Program, který je používán pro výměnu zpráv se nazývá instant messenger. Díky rozšířenosti internetu a technologické úrovni mobilních telefonů se „instant messangery“ přesunuly z desktopů i do přenosných zařízení, kde částečně nahrazují (nebo spíše doplňují) poměrně drahé SMS.

Instant messaging (dále IM) je v současné době jednou z nejrozšířenějších forem komunikace. Jedná se o zasílání textových zpráv mezi dvěma klienty. K tomuto účelu v dnešní době existuje celá řada protokolů a klientů. Mezi nejznámější patří AIM, Yahoo, Skype, ICQ a Jabber. Poslední zmíněný protokol (Jabber) je nejrozšířenější a jeho hlavní výhodou je, že se jedná o open-source[2] software, díky čemuž lze libovolně vytvářet nové klienty.

Aplikace je zaměřena na platformu Android od společnosti Google, která se v posledních letech dostala do popředí. Jedná se o open-source platformu se širokou škálou aplikací. Velmi důležitým faktorem je zejména podpora pro práci se senzory mobilního zařízení.

1.1 Cíl bakalářské práce

Cílem bakalářské práce je vytvoření klienta na protokolu XMPP. Klient nejenže umožní výměnu zpráv s některým z kontaktů, ale zároveň do něj umožní zapojit vědomí o aktuálním kontextu. K získání povědomí jsou vyžívány mobilní senzory. Kontext poslouží k nastavení „stavu“, tedy například (DND, Away, atd…), případně jiných nastavení klienta.

1.2 Stručný popis kapitol

Následující kapitola je věnována popisu protokolu XMPP/Jabber, Androidu, mobilních senzorů, SQLite databáze, klasifikátorů se zaměřením na klasifikátor Naive Bayes a v neposlední řadě také stručný popis existujících řešení. Ve třetí kapitole je popsán návrh aplikace, zejména její očekávaná funkčnost, architektura sběru dat, komunikace se serverem a struktura databáze. Poté následuje kapitola popisující samotnou implementaci aplikace s ukázkami zdrojového kódu aplikace. Posledními kapitolami jsou záve, seznam požité literatury a seznam příloh.

[1] Zasílání zpráv v téměř reálném čase
[2] Software s otevřeným zdrojovým kódem

2 Analýza

2.1 Protokol

Data jsou v síti odesílána ve formě paketů, kde každý paket obsahuje hlavičku a data. Protokolem je soubor pravidel, která určují způsob komunikace mezi dvěma subjekty v síti. Pravidla popisují formát hlavičky paketu, jak nakládat s pakety, postupy při ošetřování chyb atd. [1]

2.2 Extensible Messaging and Presence Protocol (XMPP)

Protokol pro posílání zpráv a zjištění stavu, původně vznikl jako protokol pro IM síť Jabber. Brzy se ale ukázalo, že kromě IM může být s výhodou použit i pro vzájemnou komunikaci programů nebo pro ovládání různých automatických služeb. Později byl adoptován jakožto standard Internetu do RFC dokumentů - základní normy jsou RFC 3920 (obecná specifikace protokolu) a RFC 3921 (samotný instant messaging a zobrazení stavu). RFC obsahující některá další rozšíření XMPP protokolu jsou například RFC 3922 a RFC 3923. O vývoj protokolu se stará XMPP Standards Foundation. Rozšíření nad rámec RFC jsou vydávána v podobě tzv. XEP (XMPP Extension Protocol), kterých je v současné době kolem dvou set (v různém stavu, od prvních návrhů až po standardy). XMPP je implementací obecného značkovacího jazyka XML. [1]

Architektura XMPP je decentralizovaná, díky čemuž výpadek jednoho serveru nemá nikterak velký dopad na fungování celé sítě (u většiny ostatních IM protokolů, které jsou centralizované, znamená znepřístupnění celé sítě). Seznam kontaktů je uložen na serveru (jde o klient-server architekturu), což zajišťuje dobrou přenositelnost mezi více zařízeními. Uživatel tedy uvidí všude stejné kontakty.

Každý uživatel je v síti identifikován jedinečným číslem JID (Jabber ID). Kombinace **uživatel** a **server** je unikátní (to znamená, že **uživatel** je unikátní v rámci jednoho serveru). Formát JID je podobný emailové adrese, tedy **uživatel@server**. Účet je možné dále rozdělit, např. **uživatel@server/pc1** a **uživatel@server/pc2**, což se využívá pro přihlášení uživatele z více klientů zároveň. Rozhodnutí, kam zprávu doručit, probíhá buď na základě priority, nebo celé adresy.

O komunikaci mezi uživateli se starají servery, u kterých mají daní uživatelé založený účet. Při odesílání klient odešle zprávu svému serveru. V případě, že je cílový server dostupný, předá zprávu cílovému serveru, který se postará o doručení konkrétnímu klientovi, v opačném případě si zprávu uchová server odesílatele. Jestliže je cílový uživatel nedostupný, je zpráva uchována na serveru a doručena až když se uživatel připojí.

Komunikace je zajištěna pomocí TCP (Transmission Control Protocol), přičemž výměnu dat začíná klient, který se připojí na port serveru 5222. Komunikace mezi jednotlivými servery probíhá na portech 5269.

2.2.1 Roster

Seznam kontaktů je uložený na serveru, díky čemuž je dostupný a stejný pro každého klienta daného uživatele. Roaster uchovává informace o stavu a skupině každého kontaktu, přičemž toto je reprezentováno pomocí elementu <item> s atributy JID a SUBSCRIPTION, povolené hodnoty jsou: [13]

None - neinformují se
To - uživatel je podepsán do presence kontaktu
From - kontakt je podepsán do presence uživatele
Both - navzájem se informují

2.2.2 XML Stream

XML Stream slouží k výměně elementů mezi dvěma klienty. Jelikož je stream jednosměrná komunikace, jsou pro správné fungování zapotřebí dva streamy (pro každého klienta jeden). Stream je implementován pomocí XML, jako hlavní značky se používa <stream></stream>, do kterého jsou vloženy XML stanzy (viz níže) a nastavení streamu. [13]

Atributy streamu:

- To - JID cíle
- From - JID odesílatele
- Id - Identifikátor
- Xml:lang - Jazyk
- Version - Verze

2.2.3 XML Stanza

Vlastní zasílaní zpráv a změny stavů/statusů jsou v XTMPP implementovány pomocí jazyka XML, kde existují tři druhy elementů. [13]

- <presence/> - Slouží k vyřízení připojení, autorizace a statusů.
- <message/> - Slouží k odesílání zpráv. Obsahuje elementy nižší úrovně.
  - subject - Obsahuje titulek zprávy.
  - body - Vlastní tělo zprávy.
  - thread - Vlákno, na jehož základě identifikujeme zprávu.
- <iq/> - Info/Query, jde o informace, požadavky a odpovědi. Obsahuje pod-elementy.
  - get - Požadavek na data.
  - set - Požadavek na nastavení dat.
  - result - Odpověď na get/set.
  - error - Informace o chybě.

Presence Stanza může obsahovat informace o stavu klienta, kde možné hodnoty jsou:

- Available
- Unavailable
- Subscribe

- Unsubscribe
- Subscribed
- Unsubscribed
- Probe
- Error

Stav klienta, který je dostupný (Available), může být rozšířen pomocí elementů: [13]

<show>
- Chat - připraven k chatu
- Away - nepřítomný
- Dnd - nerušit
- Xa - dlouhou dobu nepřítomný

<priority> - defaultně nastavena na 0, jde o hodnotu z intervalu <-128, 127>, vyšší hodnota vyjadřuje vyšší prioritu
<status> - může obsahovat text

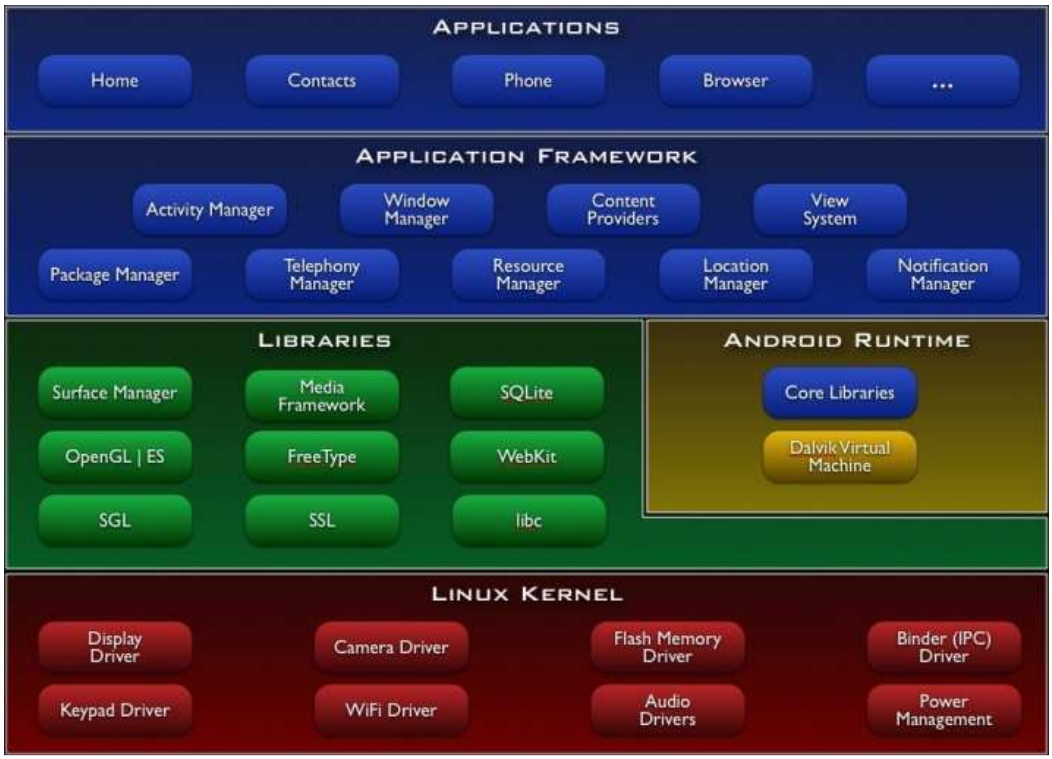
2.3 Android

Android je rozsáhlá open-source platforma, která vznikla zejména pro mobilní zařízení (chytré telefony, PDA, navigace, tablety). Zahrnuje v sobě operační systém (založený na jádru Linux), middleware, uživatelské rozhraní a aplikace. Vyvíjí ho konsorcium Open Handset Alliance, jehož cílem je progresivní rozvoj mobilních technologií, které budou mít výrazně nižší náklady na vývoj a distribuci, a zároveň spotřebitelům přinese inovativní uživatelsky přívětivé prostředí. Při vývoji systému byla brána v úvahu omezení, kterými disponují klasické mobilní zařízení jako výdrž baterie, menší výkonnost a málo dostupné paměti. Zároveň bylo jádro Androidu navrženo pro běh na různém hardwaru. Systém tak může být použit bez ohledu na použitý chipset, velikost či rozlišení obrazovky.[2]

2.3.1 Architektura

Operační systém Android se skládá z pěti komponent a jako základ bylo použito jádro Linuxu. Přestože je kód aplikací psán v Javě nevyužívá Android klasický bytecode[3], ale speciální Dalvik[4]
, který běží pod Dalvik Virtual Machine, což je virtuální stroj podobný Java Virtual Machine, oproti němuž je optimalizovaný pro běh na mobilních zařízeních. [2]

[3] Kód, do kterého překládány programy na Java Virtual Machine.
[4] Dalvik je program, ve kterém běží aplikace na zařízeních s platformou Android



Obrázek č. 1: Architekura platformy Android [2]

2.3.1.1 Jednotlivé vrstvy

Applications

- Jedná se o aplikace, které využívá uživatel. Jsou zde zahrnuty aplikace vestavěné (email, SMS, přehrávač, atd.), ale i aplikace třetích stran. Tyto aplikace jsou napsány v jazyce Java, což umožňuje rychlý vývoj a přenositelnost kódu mezi různými zařízeními. [3]

Application Framework

- Tato vrstva zpřístupňuje aplikacím data a služby systému (např. senzory mobilního zařízení), což dělá vrstvu důležitou pro vývojáře aplikací.
- Views Systém – sestavení uživatelského rozhraní aplikace, obsahuje seznamy, gridy, textová pole, atd.
- Content Providers – zprostředkovává přístup k datům jiných aplikací.
- Resource Manager – přístup k souborům designu, grafice, atd.
- Notification Manager – zobrazení upozornění ve stavovém řádku.
- Activity Manager – Spravuje životní cyklus aplikací. [3]

Libraries

- Jde o knihovny napsané v C/C++, které jsou zpřístupněny pomocí Application Framework.
- SGL – 2D grafický engine.
- Media Framework – podpora pro přehrávání audio/video souborů.
- FreeType – Rendering fontů.
- SSL
- SQLite – Relační databázová vrstva.
- WebKit – Webový prohlížeč.
- Libc – Standardní knihovna C. [4]

Android Runtime

- Virtuální stroj (podobný Java Virtual Machine), který využívá vlastnosti linuxového jádra (práce s vlákny, správa paměti) a který je optimalizovaný na provoz na mobilních zařízeních (výkon, úspora energie). Překládá Java bytecode na Dalvik bytecode.

Linux Kernel

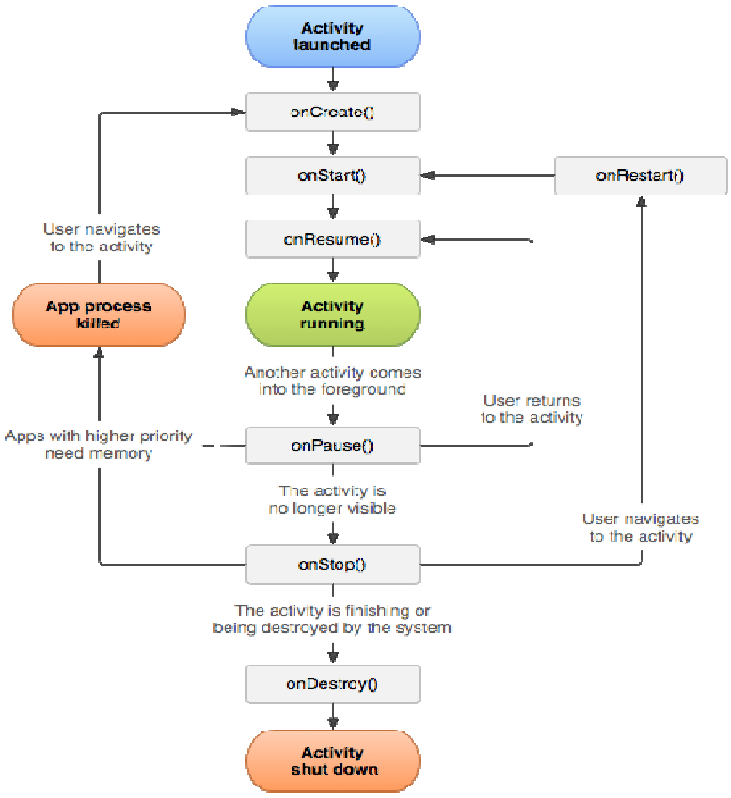
- Základem jádra je Linux 3.0, ze kterého využívá zejména správu sítí, paměti a procesů. Každá aplikace běží jako samostatný proces. [4]

2.3.2 Aktivita a její životní cyklus

Aktivita je komponenta, která představuje obrazovku s ovládacími prvky. S aktivitou může uživatel pracovat interaktivním způsobem (jako například napsat SMS, pořídit fotografii). Typicky se aplikace skládá z několika aktivit, kde jedna bývá hlavní (spuštěná po startu aplikace). Při startu nové aktivity je předchozí aktivita zastavena a nová umístěna do zásobníku aktivit. Ve chvíli, kdy je práce s novou aktivitou ukončena a je stisknuto tlačítko „Zpět“, je tato aktivita vybrána z vrcholu zásobníku a následně odstraněna, přičemž aktivní se stává aktivita předchozí. [5]

Aktivita během svého života prochází stavy onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart(), onDestroy(). Posloupnost přechodů lze vyčíst z obrázku č. 2.

Celý život - Probíhá mezi voláním onCreate() a onDestroy().
Viditelný život - Probíhá mezi voláním onStart() a onStop(). Čas, kdy může uživatel s aktivitou pracovat.
Život v popředí - Probíhá mezi voláním onResume() a onPause().



Obrázek č. 2: Životní cyklus aktivity [3]

2.4 Mobilní senzory

Dnešní mobilní telefony obsahují celou řadu senzorů, jejichž dostupnost na jednotlivých verzích Androidu zobrazuje tabulka (tabulka č. 1). S jejich pomocí může zařízení „vnímat“, co se děje v jeho okolí a podle toho přizpůsobit svá nastavení (např. zesílení/ztlumení podsvícení displeje) nebo jinak zareagovat.

Senzory v mobilních zařízeních se dělí na dva typy – hardwarové a softwarové. Hardwarové senzory jsou fyzické komponenty zabudované např. v mobilním telefonu, které samy měří nějakou fyzikální vlastnost. Oproti tomu softwarové senzory získávají data od hardwarových senzorů a provádí s nimi výpočty. [6]

Zde jsou popsány senzory podporované systémem Android.

- Akcelerometr - Měří zrychlení v osách x, y, z. Započítává gravitační sílu.
- Okolní teplota - Měří okolní teplotu ve stupních Celsia.
- Gravitace - Měří gravitační sílu v osách x, y, z.
- Gyroskop - Měří úroveň otáčení zařízení.
- Světlo - Měří úroveň okolního světla.
- Lin. akcelerace - Měří zrychlení v osách x, y, z,. Nezapočítává gravitační sílu.
- Magnetické pole - Měří úroveň magnetického pole v okolí telefonu.
- Orientace - Měří otočení zařízení.
- Tlak - Měří atmosférický tlak v okolí.
- Vzdálenost - Měří vzdálenost od objektu.
- Vlhkost - Měří vlhkost okolního vzduchu.
- Rotační vektor - Měří orientaci zařízení.
- Teplota - Měří teplotu zařízení.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a	n/a
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes	Yes	Yes	Yes
TYPE_PRESSURE	Yes	Yes	n/a	n/a
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes	Yes	Yes	Yes

Tabulka č. 1: Podpora senzorů na různých verzích Androidu

2.5 Klasifikátory

Jedná se o formu strojového učení (učení s učitelem), kde klasifikátor je algoritmus, který je při vhodné množině znalostí schopen rozdělovat vstupní data s hodnotami atributů (příznaků), do výstupních (předem zvolených) skupin. Algoritmus se učí charakteristiku jednotlivých tříd na trénovací množině dat [5]. [7]

Pokud máme množinu dat $D = \{d_1, \ldots, d_n\}$ a množinu tříd $T = \{t_1, \ldots, t_m\}$, lze klasifikaci matematicky vyjádřit jako zobrazení $D \rightarrow T$, kde je pro každé d_i definována právě jedna třída. Třída t_j obsahuje právě ty prvky, které se pomocí funkce f zobrazí do této třídy, tedy $t_j = \{d_i \mid f(d_i) = t_j, pro\ všechna\ d_i \in D\}$. [8]

2.5.1 Druhy klasifikatorů

2.5.1.1 Rozdělení dle použitých metod klasifikace

- **Symbolické** - Metody založené na rozhodovacích stromech (např ID3).
- **Subsymbolické** - Biologicky inspirované metody (např. neuronové sítě).
- **Statistické** - Využívající statistické metody (např. Naive Bayes).
- **Paměťové** - Založené na ukládání instancí tříd (např. IBL).

2.5.1.2 Rozdělení dle charakteru učení

- **Dávkové** - Zpracuje celou cvičnou množinu naráz. Typické pro symbolické metody klasifikace.
- **Inkrementální** - Cvíčné příklady lze dodat postupně, znalost se podle nich aktualizuje. Typické pro statistické metody.
- **Inkrementální se zapomínáním** - Zapomínání části znalostí může být výhodné, pokud byl některý významný atribut skryt. [7]

2.5.2 Naive Bayes classifier

„*Jde o jednoduchý pravděpodobnostní klasifikátor, který je založený na Bayesově teorému se silným předpokladem nezávislosti. Klasifikátor předpokládá, že přítomnost nebo nepřítomnost nějakého atributu je nezávislá na přítomnosti nebo nepřítomnosti jiného atributu v rámci určité třídy (z čehož je také odvozen název).*“ [9]

„*I přes svou jednoduchost je mnohdy klasifikátor velmi efektivní (hlavně při práci s rozsáhlým množstvím dat) i ve srovnání s mnohem složitějšími klasifikátory. Mezi nesporné výhody Naive Bayes klasifikátoru patří zejména malá náročnost na velikost trénovacích dat, které jsou potřebné pro učení klasifikátoru – díky nezávislosti prvků je potřeba zjistit pouze rozptyl a nikoli celá kovarianční matice. Při výsledné klasifikaci je zvolena třída s nejvyšší pravděpodobností.*“ [9]

[5] Množina dat, pro které je známá jejich příslušnost ke skupinám

2.5.2.1 Pravděpodobnostní model

„Nechť T je trénovací množina zařazení do tříd. Každý prvek z trénovací množiny je reprezentovaný n-rozměrným vektorem vlastností $X = (x_1, x_2, \dots, x_n)$ *patřícím n atributům* $A_1, A_2, \dots, A_n$. *Mějme m tříd* $C_1, C_2, \dots, C_m$. *Neznámý příklad X bude klasifikovaný do třídy s největší posteriorní pravděpodobností* $P(X \mid C_i) > P(C_i \mid X)\ pro\ 1 \leq j \leq m, j \neq i$.

Pokud vycházíme z Bayesova teorému $P(C_i \mid X) = \frac{P(X \mid C_i) * P(C_i)}{P(X)}$ *, kde* $P(X)$ *je konstantní pro všechny třídy* C_i, *stačí tedy počítat výraz* $P(X \mid C_i) * P(C_i)$, *kde* $P(C_i)$ *vyjadřuje pravděpodobnost, s jakou libovolný prvek patří do třídy* C_i. *Při použití vztahu* $P(C_i) = \frac{S_i}{S}$ *, kde* S_i *představuje počet příkladů třídy* C_i, *S představuje počet všech příkladů. Díky nezávislosti atributů při příslušnosti k dané třídě můžeme určit pravděpodobnost* $P(x_1 \mid C_1)\ x\ (x_2 \mid C_2)\ x \dots (x_n \mid C_n)$. *Pokud je* A_k *kategorický atribut, pak* $P(x_k \mid C_i) = \frac{S_{ik}}{S}$ *, kde* S_{ik} *je počet příkladové množiny patřících třídě* C_i, *pro které* $A_k = x_k$ *je počet vzorků z trénovací množiny patřící třídě* C_i. *Pro spojité atributy* A_k *se předpokládá Gaussovo normální rozložení, pak* $P(x_k \mid C_i) = g(x_k, \mu_{Ci}, \sigma_{Ci}) = \frac{1}{\sqrt{2\pi}\sigma_{Ci}} e^{-\frac{(X-\mu_{Ci})^2}{2\sigma_{Ci}^2}}$ *, kde* μ_{Ci} *je střední hodnota a* σ_{Ci} *je rozptyl hodnot atributu* A_k*trénovacích příkladů z třídy* C_i.

Pří klasifikaci je vypočten $P(X \mid C_i)\, P(C_i)$ *pro každou třídu* C_i. *X je poté přiřazeno do je* $P(X \mid C_i)\, P(C_i)$ *nejvyšší.*“ [10]

2.5.2.2 Příklad

Klasifikace osoby, zda se jedná o muže nebo ženu, na základě výšky, váhy a velikosti nohy.

Tréninková data

pohlaví	výška (m)	hmotnost (libry)	velikost nohy (mm)
muž	6	180	12
muž	5.92	190	11
muž	5.58	170	12
muž	5.92	165	10
žena	5	100	6
žena	5.5	150	8
žena	5.42	130	7
žena	5.75	150	9

Tabulka č. 2: Trénovací data klasifikátoru

Klasifikátor vytvořil pomocí Gaussova rozložení data použitelná pro klasifikaci.

sex	průměr (výška)	rozptyl (výška)	průměr (hmotnost)	rozptyl (hmotnost)	průměr (velikost nohy)	rozptyl (velikost nohy)
muž	5,855	3.5033e-02	176,25	1.2292e 02	11.25	9.1667e-01

žena	5,4175	9.7225e-02	132,5	5.5833e 02	7.5	1.6667e 00

Tabulka č. 3: Data pro klasifikaci

Za předpokladu, že pravděpodobnost ženy a muže je stejná (0,5), může klasifikátor přiřadit osobu do třídy muž nebo žena.

Testovaná data

pohlaví	**výška (stop)**	**hmotnost (libry)**	**velikost nohy (mm)**
vzorek	6	130	8

Tabulku č. 4: Vzorek dat ze zpracovaní klasifikátorem

$$Posterior(muž) = P(muž)\ p(výška|muž)\ p(hmotnost|muž)p(velikost\ nohy|muž)$$
$$Posterior(žena) = P(žena)\ p(výška|žena)\ p(hmotnost|žena)p(velikost\ nohy|žena)$$
$$P(výška|muž) = \frac{1}{\sqrt{2\pi\sigma^2}}\ e^{\frac{-(6-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi * 3.5033\ *\ 10^{-2}}}\ 10^{\frac{-(6-5.885)^2}{2*3.5033*10^{-2}}} \approx 1.5789$$
$$p(váaha|muž) = 5.9981\ *\ 10^{-6}$$
$$p(velikost\ nohy|muž) = 1.322 * 10^{-3}$$
$$Posterior(muž) =\ 6.1984 * 10^{-9}$$

$$p(výška|žena) = 2.2346 * 10^{-1}$$
$$p(váaha|muž) = 1.3112 * 10^{-2}$$
$$p(velikost\ nohy|muž) =\ 2.8669\ * 10^{-1}$$
$$Posterior(žena) =\ 5.3778 * 10^{-4}$$

Jelikož je Posterior(žena) větší než Posterior(muž), bude osoba klasifikována jako žena. [9]

2.6 SQLite databáze

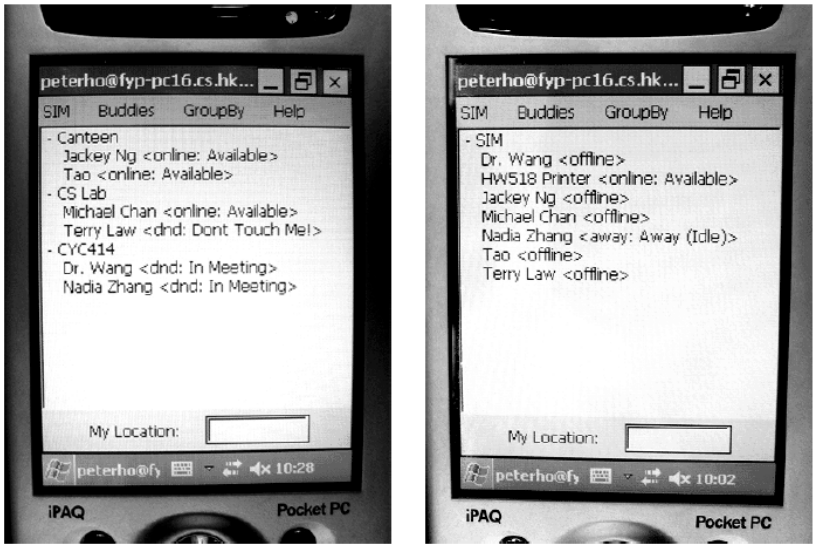
Jak již název napovídá, jedná se o odlehčenou verzi lokálního relačního databázového serveru pro Android. Přesněji nejde o server, ale pouze knihovnu napsanou v jazyce C, kterou lze jednoduchým způsobem připojit k programu. SQLite je multi-platformní [6]. Jednotlivé databáze jsou uloženy ve zvláštním souboru (přípona .dbm). Tato databáze obsahuje veškeré základní mechaniky relačních databází, což plně postačuje potřebám aplikací, které nejsou přímo zaměřeny na práci s databázovými daty.

[6] Funguje na více platformách

2.7 Existující řešení

2.7.1 Smart Instant Messenger @ HKU

Klient dostupný na platformě Windows Mobile. Design je velmi jednoduchý, což odpovídá době vydání (rok 2005) a platformě. Jedná se o produkt výzkumné skupiny University of Hong Kong. Messenger je zaměřen zejména na dynamickou organizaci kontaktů. Existují zde dva módy řazení kontaktů. Prvním je řazení podle aktivity uživatele, to vytvoří skupiny kontaktů se stejnou aktivitou. Druhou možností je řazení podle polohy kontaktů. Nejedná se však pouze o klienta, ale musí existovat i uzpůsobený server, jelikož toto řešení rozšiřuje XMPP protokol.



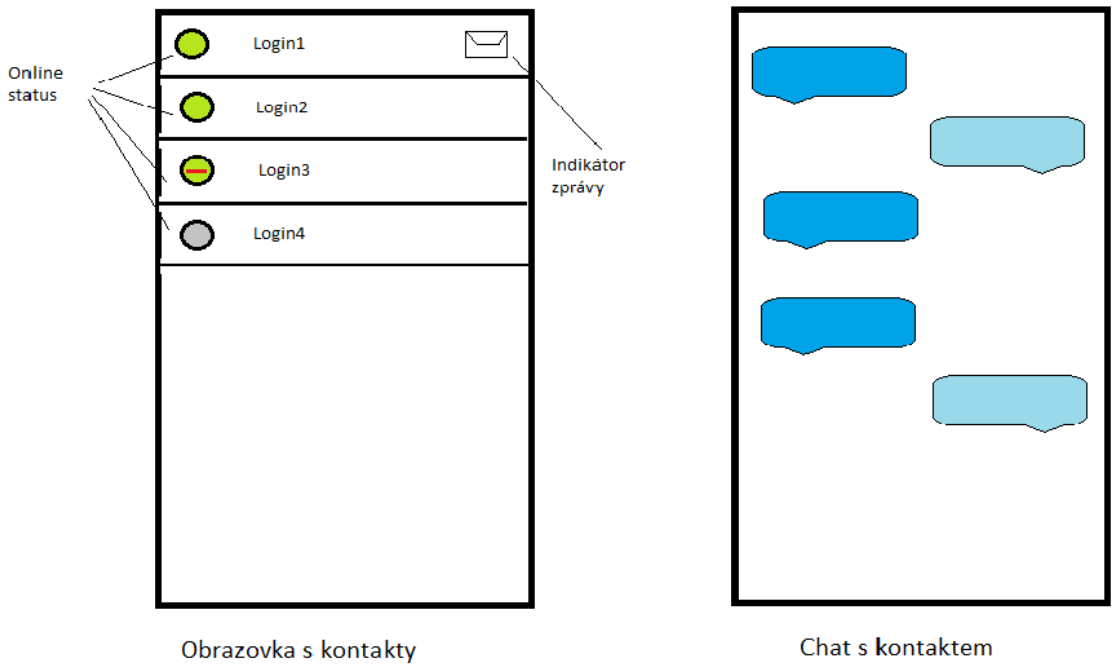
Obrázek č. 3: Smart Instant Messenger @ HKU [12]

3 Návrh aplikace

Cílem aplikace je vytvoření Jabber klienta pro mobilní platformu Android. Oproti běžným klientům by měl navíc využívat informace získané pomocí senzorů mobilního zařízení, které využije k nastavení stavu, automatické odpovědi nebo jiné konfiguraci/akci.

3.1 Uživatelské rozhraní

Jelikož se jedná o aplikaci pro mobilní zařízení, která mají v dnešní době průměrnou úhlopříčku mezi 3.0 a 4.5 palci, je nutné vytvořit dostatečně přehledné a hlavně jednoduché ovládání, aby nedocházelo k situaci, kdy uživatel nemůže kliknout některou volbu/ikonu.



Obrázek č. 4: Návrh vzhledu aplikace

Hlavní obrazovka po přihlášení obsahuje seznam kontaktů. Po vyvolaní nastavení lze u kontaktů vybrat, zda chce uživatel zobrazit online, offline kontakty, případně ponechat defaultní hodnotu „vše“, kdy v seznamu zůstanou jak online, tak offline kontakty.

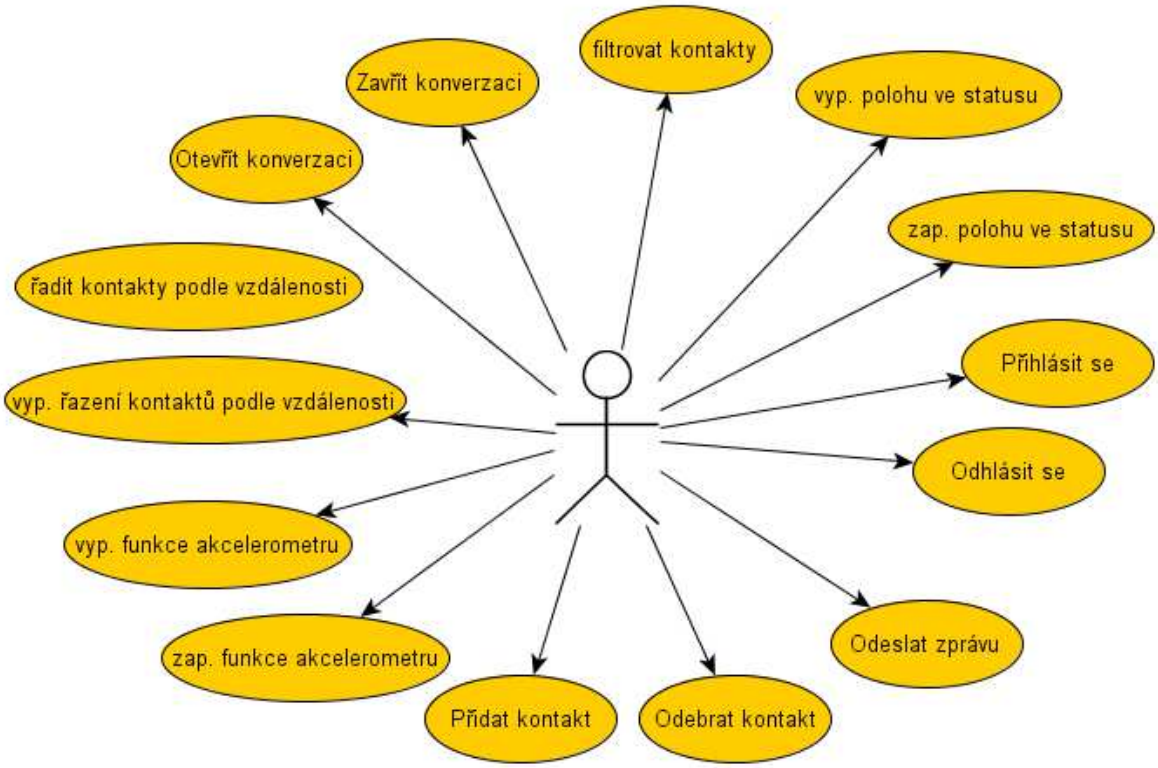
Obrazovka s menu obsahuje několik položek. **Settings** zobrazí veškerá nastavení klienta od nastavení notifikací až po automatické odpovědi a zobrazování polohy uživatele v odpovědi. Položka „**Logout**“ slouží k odhlášení uživatele. Jako poslední je zde položka „**Add new contact**“, která v reakci na zmáčknutí vyvolá dialogové okno pro přidání nového kontaktu.

3.2 Funkcionalita

Požadavky na aplikaci

- Přidání, smazání, modifikace kontaktu
- Zasílání zpráv a přijímaní zpráv
- Upozornění na zprávu od uživatele, s nímž nemám otevřenou konverzaci
- Přehled kontaktů
- Automatizovaná změna stavu

Uživatel může provádět různé akce, nejdůležitější z nich zobrazuje diagram užití.



Obrázek č. 5: Diagram případů užití

3.2.1 Komunikace se serverem

Pro komunikaci se serverem Jabber existuje pro jazyk Java několik knihoven, z nichž velmi zajímavou je knihovna Smack. S touto částí uživatel pracuje pouze okrajově přes grafické rozhraní akcí „Přihlásit se“ a „Odhlásit se“, čtením/zasíláním zpráv a nastavením stavu.

3.2.1.1 Smack

Open-source knihovna napsaná v Javě, která pracuje s XMPP protokolem. Knihovna má propracovanou dokumentaci. Mezi její nesporné výhody patří odstínění aplikačního programátora od síťové komunikace. Knihovna implementuje řadu listernerů [7], správu rosteru, prezence a chatů.

3.2.2 Povědomí o kontextu

Application Framework umožňuje přístup k senzorům mobilního zařízení. Takto získaná data aplikace může využít k nastavení stavu, či jiné konfiguraci nebo automatickým odpovědím.

Ukázky možných stavů.
Chat – Uživatel drží telefon v ruce a píše zprávu.
Away – Telefon leží na stole.
Away – Uživatel má telefon v kapse kalhot a chodí.

Aplikace umí rozpoznat jedno gesto – rychle zatřepání ze strany na stranu. Pokud se aplikace nachází ve stavu psaní zprávy a rozpozná toto gesto, smaže rozepsanou zprávu. Mazání delších zpráv je jinak časově náročnější a otravnější. Aplikace ovšem zareaguje pouze na rychlé zatřepání, aby nedošlo ke zbytečným a nechtěným vymazáním zprávy.

3.2.2.1 Architektura sběru dat

Aplikace bude využívat zejména akcelerometru pro rozpoznávání aktivity uživatele. Akcelerometr měří zrychlení v souřadnicích x, y a z. Data jsou přijímána jako tří-hodnotový vektor. Jelikož na všechny objekty na Zemi působí zemská gravitační síla, zaznamenává ji i akcelerometr. Tedy mobilní zařízení položené na podložce v klidovém stavu vykazuje akceleraci směrem ke středu Země. Vektor akcelerace ovšem míří vzhůru (pozitivní hodnota souřadnice z), i když je přitahován směrem dolů (tudíž zrychlení by bylo nulové pouze při volném pádu). V klidovém stavu na rovném povrchu totiž neměří přitahování směrem do středu Země, ale sílu, kterou působí podložka na mobilní zařízení.

Čistá data naměřená akcelerometrem obsahují šum, je potřeba použít filtr, který hluk odstraní nebo alespoň zredukuje. Pro filtrovaní bude použit low-pass a high-pass filtr.

Low-pass filtr je vyhlazovací funkce, která způsobí, že je výsledný signál hladší a méně závislý na náhlých změnách. Jelikož poloha akcelerometru ovlivňuje rozpoznávání aktivit, je nutné odstranit vliv gravitace, což pro nás udělá právě low-pass filtr.

Je také možné vyfiltrovat signál tak, že nízko frekvenční data jsou zredukována a vysokofrekvenční jsou nedotčena. Filtrace toho typu lze dosáhnout pomocí high-pass filtru – pomáhá odstranit vliv gravitace a brát v potaz pouze rychlé změny akcelerace.

Algaritmus low a high-pass filitru využívá nízkých hodnot filtrovacího faktoru. Je využíváno 20 % nefiltrovaných dat a 80 % již vyfiltrovaných dat. Současná data jsou uložena v Current, předchozí v Prev.

alpha=0.8
Low_X = alpha × Prev_X + (1 − alpha) × Current_X

[7] Listener naslouchá událostem, např. změna stavu uživatele

Low_Y = alpha × Prev_Y + (1 − alpha) × Current_Y
Low_Z = alpha × Prev_Z + (1 − alpha) × Current_Z
High_X = Current_X − Low_X
High_Y = Current_Y − Low_Y
High_Z = Current_Z − Low_Z

Následně je potřeba z vyfiltrovaných dat získat atributy, které se později použijí k vytvoření konečného vektoru atributů, který se používá pro strojové učení (klasifikaci). Atributy jsou získány za pomocí techniky zvané overlapping sliding windows (překrývající se posuvná okna).
Overlapping sliding windows jsou často používána k rozpoznávání aktivit (vzorů v datech). Algoritmy se ve většině případů nepokouší rozpoznat zvlášť data, co přijdou ze senzorů, ale data za nějaký časový interval (okno). Velikost okna je časový interval, po který jsou data sbírána (než se okno naplní). Pokud se v okně nějaké vzorky dat překrývají, jedná se o overlapping sliding widnow. Jako vzorkovací frekvence bude použito 5 Hz kvůli nízké spotřebě baterie. Velikost okna bude nastavena na 32, což při vzorkovací frekvenci 5 Hz odpovídá časovému intervalu 6,4 sekundy. Tato časová hodnota ovšem platí jen pro první okno, další okna již využívají překrývaní. Nastavením překrývání na 50 % (které je známo jako nejvhodnější) dosáhneme 3,2 sekundy na jedno okno, což je doba, za kterou je stále možné rozpoznat jednotlivé aktivity.
Pro každou složku vektoru zrychlení (tedy x, y, z) bude vypočítán průměr, energie, standardní odchylka průměru a standardní odchylka energie, což nám v konečném součtu dává 12 atributů pro vzorek dat.
Pro výpočet průměrné hodnoty zrychlení v ose X použijeme vzorec $meanX = \frac{\sum_{i=0}^{WINDOW\ SIZE} window[i].X}{WINDOW\ SIZE}$, kde WINDOW_SIZE označuje velikost okna (tedy 32), $window[i]$ představuje úložiště dat celého jednoho okna a $window[i].X$ hodnotu zrychlení v rozmezí 0 až WINOW_SIZE, obdobně pro průměrné zrychlení v osách Y, Z. Průměrná energie v ose X je vypočítána obdobně jako průměrné zrychlení, tedy $energyMeanX = \frac{\sum_{i=0}^{WINDOW\ SIZE} window[i].EnergyX}{WINDOW\ SIZE}$, kde $window[i].EnergyX$ představuje hodnotu zrychlení v rozmezí 0 až WINDOW_SIZE, stejně tak pro průměrnou energii v osách Y, Z. Standardní odchylka zrychlení od průměrné hodnoty v jednotlivých osách se vypočte pomocí vzorce $meanStandardDeviationX = \sqrt{\frac{\sum_{i=0}^{WINDOW\_SIZE}(window[i].X - meanX)^2}{WINDOW\_SIZE}}$, kde $window[i].X$ je hodnota zrychlení v rámci okna na indexu i a $meanX$průměrná hodnota zrychlení v ose X, obdobně by se standardní odchylka zrychlení vypočítala i pro zbylé osy. Standardní odchylka energie v ose X by se vypočítala takto $energyStandardDeviationX = \sqrt{\frac{\sum_{i=0}^{WINDOW\_SIZE}(window[i].EnergyX - energyMeanX)^2}{WINDOW\_SIZE}}$, kde $window[i].EnergyX$ je hodnota energie v rámci okna na indexu i a $energyMeanX$je průměrná hodnota energie v rámci okna, obdobně pro osy Y, Z.
Pro zpracování atributu bude dále použit Naive Bayes klasifikátor popsaný v předchozích sekcích. Klasifikátor má za úkol rozpoznat čtyři aktivity – třepání telefonem ze strany na stranu, chůzi, klidový stav na stole a držení mobilu v rukou v poloze běžné pro práci s telefonem. [11]

3.2.2.2 Získání přibližné polohy uživatele

Pro získání polohy uživatele existují dva způsoby, kde prvním z nich je použití GPS senzoru, který dokáže zjistit polohu zařízení téměř kdekoli s obrovskou přesností. Proč tedy nevyužít takto přesného

senzoru, když jej má v dnešní době zabudován téměř každý mobilní telefon s OS Android? Hlavním důvodem je obrovská náročnost na spotřebu baterie, která je neúnosná vzhledem k povaze aplikace. Další nevýhodou je špatná (nebo žádná) funkčnost uvnitř budov.

Další možností je využití takzvané triangulace v GSM síti. Mobilní telefon obvykle přijímá signál od více stanic operátora, jejichž poloha je známá (operátorovi), pro veřejnost jsou tyto informace bohužel skryty. Každá věž má identifikátor nazývaný cell ID, které slouží k určení přístupového bodu pro mobilní zařízení. I když jsou data o poloze cell ID veřejně nepřístupná, některé společnosti získaly vlastní databázi – například společnost Google získává data od uživatelů, kteří používají aplikaci Google Maps. Díky jejich webovému API je možné zaslat požadavek na zjištění polohy zařízení. Jako odpověď je přijata přibližná poloha zařízení (zeměpisná šířka a délka), které se dalším požadavkem mohou nechat přeložit na člověkem čitelnou polohu (ulice, město, stát) – tato technika se nazývá Reverse Geocoding. I když je tato metoda poměrně dosti nepřesná (obzvláště ve vesnických oblastech), je díky prakticky nulové zátěži na baterii a faktu, že funguje i uvnitř budov lepší variantou.

3.2.2.3 Kontakt list

Kontakt list je v klasických IM ve většině případů řazen podle abecedy nebo podle rozdělení do skupin. Díky možnosti získat přibližnou polohu uživatele pomocí cell ID a LAC, lze za předpokladu komunikace více zařízení pomocí této aplikace získat polohu ostatních uživatelů ze statusu (v případě že mají povolenu volbu zobrazení polohy ve statusu). Pro každý kontakt, pro který bude zjištěna poloha ze statusu, vypočítá aplikace přibližnou vzdálenost. Podle vypočtené vzdálenosti vzestupně uspořádá kontakty. Kontakty, u kterých nelze zjistit poloha, budou umístěny na konec kontakt listu. Vypočítání vzdálenosti bude probíhat pouze jednou za delší časový interval, aby nedocházelo ke zbytečnému přenosu dat. A samozřejmě bude možné takovéto uspořádání kontaktů podle vzdálenosti vypnout.

3.2.3 Konfigurace

Konfigurační údaje aplikace budou uloženy v adresovém prostoru označovaném jako shared preferences (sdílené preference), který obsahuje množinu prvků klíč–hodnota. Ke zde uloženým prvkům lze přistoupit z kterékoli části aplikace. V konfiguraci lze najít nastavení notifikací o příchozích zprávách (tón, vibrace), filtrace online/offline kontaktů, zobrazení polohy uživatele ve statusu a využívaní automatických nastavení statusů.

3.2.4 Databáze

Protokol XMPP sám o sobě nezaručuje ukládání historie zpráv. Existují sice externí služby, díky nimž je možné server-side archivace dosáhnout, na druhou stranu se stále jedna o mobilní aplikaci, jejíž internetové připojení ve značné míře podléhá FUP [8]. Jednalo by se tedy o zbytečně přenášená data.

[8] Fiar User Policy je omezení množství přenesených dat, které zamezuje, aby některý z uživatelů příliš zatěžoval internetové připojení

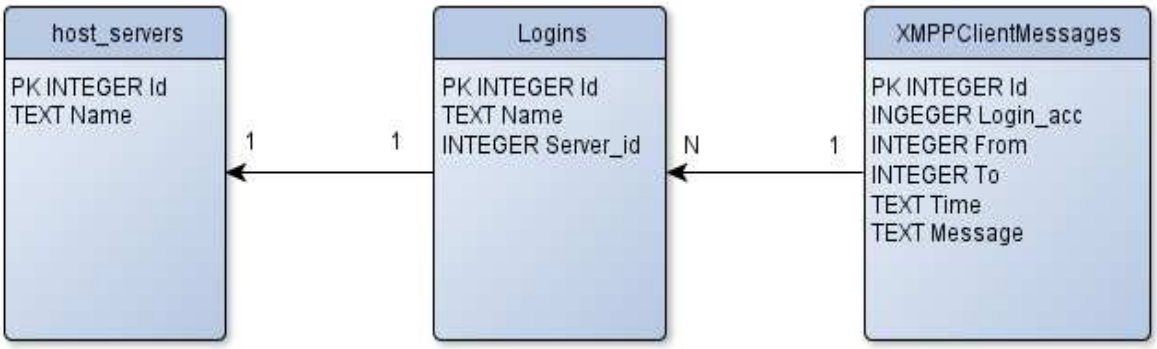
Pro realizaci klient-side historie zpráv bude aplikace využívat SQLite databáze, jejíž snadné použití plně postačuje potřebám aplikace.

3.2.4.1 Struktura databáze:

Pro ukládání zpráv pro různé účty a různé servery bude zapotřebí tří tabulek. Tabulka Host_servers bude sloužit k uložení serveru, na kterém komunikace probíhala. Toto je zapotřebí, jelikož uživatel může vlastnit účet muj_login na serverech server_1 a server_2, což by vedlo ke zmatení aplikace – ta by mohla pomíchat konverzace z různých serverů.

Další tabulkou je tabulka Logins, která bude obsahovat primární klíč, název loginu ve formě textu a id serveru, na kterém je daný login registrovaný. V této tabulce budou ukládány informace jak o odesílateli, tak o příjemci zprávy.

Hlavní tabulkou bude XMPPClientMessages, která bude obsahovat primární klíč, id loginu uživatele aplikace (přihlášeného do programu v době odeslání zprávy). Id loginu odesílatele a příjemce, které budou využity ke zpětnému dohledání zpráv (načtení části historie) a v neposlední řadě časovou značku odeslání/přijmutí zprávy a text samotné zprávy.



Obrázek č. 6: Struktura databáze

4 Implementace

Funkčnost aplikace je napsána v jazyce Java a vzhled v jazyce XML. Pro vývoj bylo využito vývojové prostředí Eclipse IDE. Aplikace se sestává ze sedmnácti tříd, všechny jsou v balíku com.example.jabbercontextawareclient, naimportované externí knihovny jsou ve složce /libs/, jmenovitě se jedná o smack.jar, smackx.jar, smackx-debug.jar, smackx-jingle.jar.

4.1 Připojení

Výchozím bodem aplikace je aktivita MainActivity.java, která se stará nabízí rozhraní pro přihlášení na některý z Jabber serverů. Aktivita obsahuje editovací pole (EditVeiw), která slouží k zadání přihlašovacího jména (jabber id bez části serveru), hesla, serveru a portu a tlačítko „Log in“, jehož stlačení obsluhuje metoda public void `onLoginPush(View w)`, která vytvoří nový objekt třídy ClientConnection.java obsluhující připojení k serveru.
Třída ClientConnection.java bere jako parametr konstruktoru login, heslo, server a port. Zde již využije třídy ze Smack API a vytvoří konfigurační soubor pro připojení. A podle typu serveru nastaví typ autentizace a komprese.

```
ConnectionConfiguration connectionConfig = new
ConnectionConfiguration(host, Integer.parseInt(port), service);
connectionConfig.setCompressionEnabled(true);
connectionConfig.setSASLAuthenticationEnabled(false);
```

Dále vytvoří objekt obsluhující samotné připojení a pokusí se připojit.

```
// connect to the server
connection = new XMPPConnection(connectionConfig);
try{
     connection.connect();
} catch (XMPPException ex){
     Log.e("XMPPConnection", "Connecting failed: " +
ex.toString());
}

// login on the server
try{
     connection.login(login, password);
} catch (XMPPException ex){
     Log.i("XMPPConnection", "Loggin failed");
}
```

Pokud se připojení podaří, vytvoří se objekt třídy DatabaseHandler.java na nejž se spolu se samotným objektem třídy ClientConnection nastaví reference do globálního objektu XMPPConn (používá Singleton[9] pattern). A dále je vytvořen litener pro příchozí zprávy, ve kterém probíhá obsluha spojení navázaných od jiných uživatelů (chatů, které nebyly vytvořeny lokálně).

```
connection.getChatManager().addChatListener(new
ChatManagerListener()
{
```

[9] Návrhový vzor, který zajistí, že daná třída bude mít pouze jednu instanci

```
	public void chatCreated(final Chat chat, final boolean
createdLocally)
	{
		if(!createdLocally)
		{
		}
	}
});
```

Následně se přepne kontext na aktivitu ContactScreenActivity.java, která slouží jako hlavní obrazovka po připojení, na této obrazovce uživatel vidí prezenci svých kontaktů. Tato aktivita rozšiřuje ListView, které je zároveň jediným obsahem této obrazovky. V metodě `onResume()` se zjišťuje konfigurace aplikace a to z důvodu, životního cyklu aktivity, která se vytvoří pouze jednou a při přepnutí do aktivity s nastavením se pouze pozastaví. Pokud by tedy načtení nastavení bylo provedeno pouze v metodě `onCreate()`, nebylo by nikdy aktuální. Samotné nastavení je zjištěno pomocí PreferenceManageru.

```
useSensorStatuses =
sharedPreferences.getBoolean(USE_SENSOR_STATUSES, true);
showPositionInStatus =
sharedPreferences.getBoolean(SHOW_POSITION_IN_STATUS, true);
```

Pokud je povoleno nastavení `USE_SENSOR_STATUSES` (povolení využívat sběru dat ze senzorů),je nastaven litener na akcelerometr s intervalem sběru dat `SENSOR_DELAY_NORMAL` (což odpovídá zhruba 200 ms).

```
sensorManager.registerListener(movementDetector, accelerSensor,
SensorManager.SENSOR_DELAY_NORMAL);
```

4.2 Sběr dat

Objekt movementDetector je instancí třídy MovementDetector.java, která obstarává obsluhu akcelerometru. Tato třída implementuje SensorsEventListener a musí v ní být implementováno několik metod, z nich hlavní je metoda public `void onSensorChanged(SensorEvent sEvent)`, která je zavolána pokaždé, když dostane nová data. Zde také dochází k převodu a filtrování surových dat z akcelerometru do formy použitelné pro klasifikaci, tedy vektoru atributu.
Jelikož jsou data ovlivněná gravitací a jiným šumem, je nejdříve potřeba tato data vyfiltrovat, k tomu slouží low-pass a high-pass filter. Nastavením koeficientu alpha na 0,8 je využíváno 80 % z předešle vyfiltrovaných dat a pouze 20 % dat nových.

```
final float alpha = 0.8f;
// Gravity components of x, y, and z acceleration
mGravity[X] = alpha * mGravity[X] + (1 - alpha) * sEvent.values[X];
mGravity[Y] = alpha * mGravity[Y] + (1 - alpha) * sEvent.values[Y];
mGravity[Z] = alpha * mGravity[Z] + (1 - alpha) * sEvent.values[Z];

mLinearAcceleration[X] = sEvent.values[X] - mGravity[X];
mLinearAcceleration[Y] = sEvent.values[Y] - mGravity[Y];
```

```
mLinearAcceleration[Z] = sEvent.values[Z] - mGravity[Z];
```

Data jsou postupně ukládána `do ArrayList<AccelData> accelerationData`, avšak `ArrayList` obsahuje pouze počet objektů rovnající se `WINDOW_SIZE` (což je 32). Objekty uchovávající atributy jednoho vzorku jsou znovupoužity pro nová data, aby nedocházelo ke zbytečnému znovu-vytváření nových objektů. Jelikož se jedná o opravdu jednoduchou třídu, nepovažoval jsem za nutné vytvářet metody pro komunikaci s takovýmto objektem, data jsou ponechána jako public.

```
public class AccelData {

	public float x = 0;
	public float y = 0;
	public float z = 0;

	public float eX = 0;
	public float eY = 0;
	public float eZ = 0;

	public AccelData()
	{
	}
}
```

Při prvním plnění okna je zapotřebí 32 vzorku dat, pro další se pouze data z horní poloviny `accelerationData` přesunou do dolní poloviny a jsou doplněna novými daty, čímž nám vniká 50 % Windows overlapping. Po naplnění okna jsou z dat v `accelerationData` vypočítány atributy pro každou pro zrychlení v každé ose. Na takto vypočtené atributy je použit Naive Bayes klasifikátor, který vypočítá pravděpodobnost zařazení do jednotlivých tříd (chůze, ležící na stole, práce v rukou, třepání ze stran na stranu).
Příklad výpočtu průměrné hodnoty zrychlení chůze vůči trénovacím datům, která jsou uložena v `AccelDataSample sampleWalk` (obsahuje vypočtené hodnoty atributů vzorku chůze).

```
float pMeanXWalk = (float)((1/Math.sqrt(2 * Math.PI * sampleWalk.sdX)) *
Math.exp((- Math.pow(accelMeanX1 - sampleWalk.meanX, 2))/(2 *
sampleWalk.sdX)));
```

Po vyhodnocení o kterou aktivitu se jedná, je navýšeno počítadlo dané aktivity. V případě rozpoznání aktivity třepání ze strany na stranu je, pokud se uživatel nachází v chatu s některým z kontaktů, vymazán obsah EditVeiw pro zadávání zprávy. V ostatních případech je po patnácti navýšeních vybráno počítadlo s nejvyšší hodnotou (značí nejpravděpodobnější aktivitu), je přes instanci třídy ClientConnection.java nastaven status uživatele (pokud se liší od předchozího). Tento proces trvá přibližně 17 sekund (záleží pouze na sběru dat. Jelikož tento process nemá na systému android prioritu, je možné, že se bude čas lehce lišit).

4.3 Kontakt list

Základem každého ListView je adaptér obsahující seznam prvků k zobrazení. U každého kontaktu se zobrazuje ikona podle jeho prezence (online, offline, dnd, away, chat), jabber ID /login kontaktu a indikátor zprávy (pokud existuje dosud nepřečtená). Jelikož tedy nestačil pro každý kontakt pouze jeden řetězec (a tím pádem ani klasický adaptér), bylo zapotřebí vytvořit pro ListView nový adaptér. K tomuto účelu slouží ContactsAdapter.java, jedá se o třídu rozšiřující BaseAdapter.java. Konstruktor dostává jako parametr kontext a ArrayList obsahující jednotlivé kontakty typu ContactEntry. O každém kontaktu jsou udržovány informace o jeho loginu, jabber ID, stavu a indikátoru nepřečtené zprávy. ContactsAdapter se sestává ze dvou ImageView (stav a indikátor nepřečtené zprávy) a jednoho TextView (jabber ID/login).

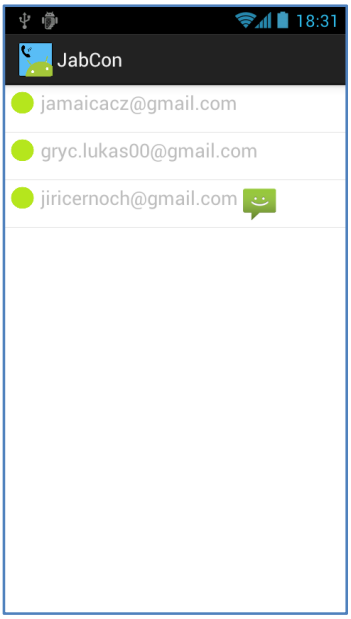
```
public class ContactEntry {
	public String m_login;
	public String m_user;
	public int m_state;
	public boolean m_messageIndicator;
	float distance;
	…
}
```

Logika změn zobrazení/skrytí indikátoru nepřečtené zprávy, změna stavu (online/offline, …) se provádí v abstraktní metodě `getView`, která jako parametr dostává pozici položky v rámci adaptéru, díky čemuž je možné přistoupit ke odpovídajícímu ContactEntry.

```
ContactEntry contact = (ContactEntry) this.getItem(position);
```

Adaptér dále pracuje s informacemi, které získá z objektu contact. TextView nastaví podle m_login/m_user, ImageView nalevo od TextView si nastaví pozadí podle m_state, které může nabývat hodnot (online, offline…) a ImageView se zobrazí/skryje podle hodnoty m_messageIndicator (defaultní hodnota je true).

```
if(contact.m_messageIndicator == false)
	viewHolder.messageIndicator.setVisibility(View.VISIBLE);
else
	viewHolder.messageIndicator.setVisibility(View.INVISIBLE);
```



Obrázek č. 7: Kontakt list

4.3.1 Řazení kontaktů

Mechanismus řazení předpokládá, že status obsahuje zprávu ve formátu „`Hi, I´m currently in MĚSTO, STÁT.`“. Za touto zprávou může následovat cokoli jiného. Status v tomto formátu je automaticky nastavován aplikací, v případě používání jiného IM je, bohužel, potřeba status nastavit manuálně. Řešení není ideální, ale při vytváření klienta není možné zasahovat do komunikačního protokolu (jak tomu je například u existujících řešení).

Pro každou položku ListView (kontakt) se aplikace pokusí získat informaci o vzdálenosti od zařízení na kterém běží. Nejprve je získán status daného kontaktu, v případě že je prázdný, a nebo neobsahuje zprávu ve výše zmíněném formátu, je pro kontakt nastavena záporná vzdálenost. Kontakty se zápornou hodnotou vzdálenosti jsou řazeny na konec kontakt listu. Pokud je vyhledání adresy ze statusu úspěšné, je adresa přeložena na zeměpisnou šířku a délku. K překladu adresy je opět využito třídy Geocoder.java.

```
geocoder = new Geocoder(context, Locale.getDefault());
List<Address> addressList = geocoder.getFromLocationName(address, 5);
Address location = addressList.get(0);
```

Následně je pro výpočet vzdálenosti použita metoda třídy Location.java, která vypočítá přibližnou vzdálenost vzdušnou čarou mezi dvěma body na Zemi.

```
Location.distanceBetween(latitude, longitude, location.getLatitude(),
location.getLongitude(), results);
```

Hodnota vzdáleností je pro každý kontakt uložena v položce `distance` instancí třídy ContactEntry.java. V případě, že je povolený tento druh řazení, jsou následně vypočtené vzdálenosti využity při vytváření ListView k seřazení jednotlivých položek.

```
Collections.sort(temp, new Comparator<ContactEntry>() {
    @Override
    public int compare(ContactEntry c1, ContactEntry c2)
```

```
    {
      if(c1.distance == -1.0f)
            return (int)c2.distance;

      if(c2.distance == -1.0f)
            return (int)c2.distance;

      return (int)(c1.distance - c2.distance);
    }
```

4.4 Práce s databází

Pro práci s databází slouží třída DatabaseHandler, která využívá SQLite a jejích metod. Z jejího konstruktoru je volána metoda `OpenDB(View w)`, která se pokusí otevřít databázi a vytvořit tabulky (pokud ještě neexistují).

```
db = w.getContext().openOrCreateDatabase("XMPPClientMessageDatabase", 0,
null);

// servers
try {
      db.execSQL("CREATE TABLE IF NOT EXISTS HostServers (id INTEGER
PRIMARY KEY AUTOINCREMENT, name TEXT);");
}catch(SQLException ex){
      Log.e("DatabaseDebug", "Cant create HostServer table");
      return false;
}
```

Třída je využívána pro ukládání zpráv a odkaz na její insanci je uložen v XMPPConn (singleton), je tedy přístupná z kterékoli části programu. Připojení k databázi zahájeno ihned po připojení uživatele a zrušeno až po jeho odhlášení, jelikož je využívána po celou dobu běhu aplikace.

4.5 Zjištení pozice

Pokud je zapnutá volba pro zobrazování pozice zařízení ve statusu, je tato služba načasována jednou za 15 minut (častější aktualizace je vzhledem k povaze aplikace poměrně zbytečná) a obsluhována instancí třídy PositionManager.java. Celá podstata této třídy tkví v získání GSM cell ID věže, ke které je zařízení připojeno a GSM LAC (Location Area Code). V konstruktoru třídy PositionManager je získána reference na TelephonyManager, který poskytuje přístup k telefonním službám zařízení. Pokud PositionManager příjme zprávu `GetCellId(Context context)`, získá GSM polohu zařízení a z ní cell ID a LAC.

```
cellLocation = (GsmCellLocation)telephonyManager.getCellLocation();
cellId = cellLocation.getCid();
LAC = cellLocation.getLac();
```

Tato data poté zformuje do http požadavku na server http://www.google.com/glm/mmap, který z cell ID a LAC vypočte přibližnou polohu zařízení v podobě zeměpisné šířky a délky, které vrátí jako odpověď.

```
DataOutputStream dataStream = new DataOutputStream(outputStream);
dataStream.writeShort(21);
dataStream.writeLong(0);
dataStream.writeUTF("en");
dataStream.writeUTF("Android");
dataStream.writeUTF("1.0");
dataStream.writeUTF("Web");
dataStream.writeByte(27);
dataStream.writeInt(0);
dataStream.writeInt(0);
dataStream.writeInt(3);
dataStream.writeUTF("");

dataStream.writeInt(cellId);
dataStream.writeInt(LAC);

dataStream.writeInt(0);
dataStream.writeInt(0);
dataStream.writeInt(0);
dataStream.writeInt(0);
dataStream.flush();
```

Zeměpisná šířka a délka ovšem není pro většinu lidí čitelná, je tedy provést ještě překlad do člověkem čitelné formy (názvy měst, státu, …). K tomu slouží metoda `TranslateToAdress(double latitude, double longitude, Context context)`, ta vytvoří instanci Androidí třídy Geocoder. Geocoder je třída, která se stará o překlad názvů ulic, měst, atd.. na zeměpisnou šířku a délku, ale také k reverznímu geocodingu, což je přesně to, co je potřeba. Geocoder se v tomto bodě využije pro reverzní překlad zeměpisné šířky a délky (metoda `getFromLocation`), které jsou získány z odpovědi od serveru http://www.google.com/glm/mmap.

```
try{ // get address lis for our latitude and longitude
      List<Address>   addressList   =   geocoder.getFromLocation(latitude,
longitude, 1);
      // check for results
      if(addressList != null && addressList.size() > 0)
      {
            adr = addressList.get(0);
            currentPosition   =   adr.getSubAdminArea()   +   ",   "   +
      adr.getCountryName();
      }
}
```

Z proměnné adr lze není získat veškeré potřebné informace ohledně polohy, jelikož ovšem není poloha přesná jako v případě využití GPS, pracuje aplikace pouze s názvem města a státu, které má uloženy ve formě stringu v proměnné currentPosition – ta je přístupna pro ostatní objekty přes metodu `GetPosition(Context context)`.

4.6 Chat

O chat se stará aktivita ChatActivity.java, která je spuštěna z ContactsScreenActivity.java v reakci na kliknutí na některý z kontaktů. ChatActivity se skáda z ListView pro zobrazení zpráv, EditView pro zadávání zprávy a tlačítka pro odeslání zprávy. ChatActivity dostane jako parametr Jabber ID uživatele, na jehož položku v ListView v ContactScreenActivity bylo kliknuto. Ihned po otevření

chatu je jako první vypsán status uživatele, se kterým je chat navázán (pokud nějaký status má) ve formě obyčejného neformátovaného textu. Pokud existuje historie, je načteno 10 posledních zpráv z konverzace. Následně je zaregistrován nový `MessageListener()`, který implementuje metodu pro zpracování příchozích zpráv a je vytvořen nový chat.

```
msgListener = new MessageListener() {
      @Override
      public void processMessage(Chat chat, Message message) {
            …
      }
}
chatWithUser = chatManager.createChat(user, msgListener);
```

V metodě zpracovávající příchozí zprávy (`processMessage(Chat chat, Message message)`) dochází k jejích archivaci do databáze a přidání do ListView. Pokud jsou povoleny zvukové notifikace, přehraje mobilní zařízení zvuk, který je nastavený v konfiguraci. V případě, že jsou povoleny i vibrace, zařízení zavibruje. Pokud není aktivita chatu s tímto uživatelem aktivní, je ke kontaktu nastaven indikátor nepřečtené zprávy.

Reakci na kliknutí tlačítka na odeslání zprávy obsluhuje metoda `DoSendMessage(View v)`, která zjistí data k odeslání z EditView a v případě, že obsahuje nějakou zprávu, pokusí se tuto zprávu odeslat. Zároveň je, stejně jako přijaté zprávy, ihned archivována v databázi a přidán do ListView, které obdrží zprávu o změně dat, což vyústí v jeho překreslení.

```
chatWithUser.sendMessage(textToSend);
listOfItems.add(new ChatMessage(textToSend, true, false, timeStamp));
dbHandler.InsertIntoMessages(myId,    myId,    chatingWithId,    timeStamp,
textToSend);
```

Zprávy jsou formátovány na základě toho, zda byly odeslány nebo přijaty. Každá zpráva se sestává ze zprávy samotné a časové značky, kdy byla odeslána/přijata. Pokud se jedná o zprávu odeslanou, je umístěna nalevo a její pozadí je modrá bublina, pro zprávy příchozí je použita tyrkysová barva bubliny a jsou umístěny napravo, což chat značně zpřehledňuje.

Každá jedna zpráva zobrazená v chatu je instancí třídy ChatMessage.java. Jsou uloženy v ArrayListu, který využívá adaptér ListView. Jelikož je zde opět potřeba dynamického formátování, je nutné vytvořit nový adaptér – tím je ChatMessagesAdapter.java.

```
listOfItems = new ArrayList<ChatMessage>();
adapter = new ChatMessagesAdapter(chat.getContext(), listOfItems);
```

Stejně jako u ContactAdapter.java, je zde veškerá logika umístěna v abstraktní metodě `getView(int position, View convertView, ViewGroup parent)`. Každá položka tohoto adaptéru obsahuje dvě TextView zarovnaná pod sebou. Pro dynamickou modifikaci rozložení je potřeba změnit LayoutParams jednotlivých zpráv a jejích částí, což se provede nastavením jejích vlastnosti gravity.

```
LinearLayout layout = (LinearLayout)
convertView.findViewById(R.id.chat_message_cover);
LayoutParams coverLayoutParams = (LayoutParams)layout.getLayoutParams();
// set gravity
coverLayoutParams.gravity = Gravity.LEFT;
```

Obrázek č. 8: Ukázka chatu

5 Závěr

Cílem práce bylo vytvořit aplikaci pro platformu Android, která bude schopna odesílat a přijímat zprávy přes protokol XMPP. Zároveň by aplikace měla využívat senzorů mobilního zařízení k získání povědomí o aktuálním kontextu.

V části práce zabývající se analýzou jsou popsány technologie, které byly, více či méně, použity při vývoji aplikace. Rozebrán je zde zejména XMPP protokol, platforma Android, senzory mobilních zařízení, klasifikační metody a existující řešení. Obdobný program se mi, bohužel, podařil najít pouze jeden a informace o něm jsou přístupné pouze ve formě krátké prezentace.

Výsledná aplikace splňuje požadavky zadané v kapitole o návrhu aplikace. Uživatelé mohou pomocí aplikace komunikovat, přidávat a odstraňovat kontakty. V nastavení lze navolit generování automatických statusů obsahující předpokládanou aktivitu uživatele (rozpozná chůzi, odložení na stole, držení v rukou). Další možností je vložení přibližné polohy uživatele do statusu, což slouží jak informativně, tak pro řazení kontaktů na základě jejich vzdálenosti od zařízení, na kterém aplikace běží. Ukládání adresy do statusu bylo voleno zejména kvůli nemožnosti měnit komunikační protokol, i když se samozřejmě nejedná o ideální řešení. Zařízení také rozpozná rychlé třepání rukou, které slouží jako gesto ke smazání aktuálně rozepsané zprávy. Aplikace udržuje historii komunikace v databázi a její část zobrazí při otevření chatu s daným kontaktem. Dále je zde možnost nastavení jak zvukové, tak vibrační notifikace příchozích zpráv.

Aplikace má základní funkcionalitu a její případný budoucí vývoj je možný zejména v oblasti využívaní a zjišťování informací o kontextu. V první řadě je možné aplikaci „naučit“ rozpoznávat mnohem více druhů aktivit, jako jsou například běh, jízda na kole a mnohé další. Přesnější rozpoznání by ovšem nejspíše vyžadovalo generování lepších atributů a nebo využití více senzoru. Dalším možným bodem budoucího vývoje je GUI, které je velmi strohé a nemůže se rovnat klasickým IM jako ICQ, Google Talk, WhatsApp, atd. Chybí zde například možnost vkládání „smajlíků“ a správy historie, tyto a mnohé další možnosti sice nebyly vyžadovány, ale jsou standardem u většiny současných IM.

6 Literatura

[1] Wikipedie: Extensible Messaging and Presence Protocol. [online]. [cit. 2013-05-13]. Dostupné z: http://en.wikipedia.org/wiki/XMPP

[2] Android(operační systém). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-13]. Dostupné z: http://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m)

[3] App Framework. [online]. [cit. 2013-05-13]. Dostupné z: http://developer.android.com/about/versions/index.html

[4] Vývoj pro Android: Architektura. [online]. [cit. 2013-05-13]. Dostupné z: http://www.elitecsoftware.cz/vyvoj-pro-android/

[5] Activities. [online]. [cit. 2013-05-13]. Dostupné z: http://developer.android.com/guide/components/activities.html

[6] Sensors Overview: Introduction to Sensors. GJORESKI, Hristijan a Matjaž GAMS. [online]. [cit. 2013-05-13]. Dostupné z: http://developer.android.com/guide/topics/sensors/sensors_overview.html

[7] Přednáška kurzu MPOV: Klasifikátory, strojové učení, automatické třídění 1. [online]. [cit. 2013-05-13]. Dostupné z: http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/09%20-%20Klasifikatory%20a%20automaticke%20trideni.pdf

[8] Data Mining: Klasifikace. [online]. [cit. 2013-05-13]. Dostupné z: http://eamos.pf.jcu.cz/amos/kat_inf/externi/kat_inf_21586/files/studijni_texty/klasifikace.pdf

[9] Naive Bayes classifier. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-13]. Dostupné z: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[10] Naive Bayes: Naivný Bayesovský klasifikátor. [online]. [cit. 2013-05-13]. Dostupné z: https://mining.fei.tuke.sk/oz/index.php/operatory-rapidminer/80-naive-bayes

[11] ACCELEROMETER DATA PREPARATION FOR ACTIVITY RECOGNITION. GJORESKI, Hristijan a Matjaž GAMS. [online]. [cit. 2013-05-13]. Dostupné z: http://dis.ijs.si/hristijan/publications/2011/IS_2011.pdf

[12] SmartInstantMessenger in Pervasive Computing Environments. [online]. [cit. 2013-05-13]. Dostupné z: http://i.cs.hku.hk/~clwang/papers/SIM_HKU_GPC2006-slide.pdf

[13] Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence . [online]. [cit. 2013-05-13]. Dostupné z: http://xmpp.org/rfcs/rfc6121.html#roster-syntax-items-subscription

7 Seznam příloh

Příloha 1. CD/DVD

7.1 Obsah přiloženého CD

- /jabbercontextawareclient/ – zdrojové soubory aplikace
- JabCon.apk – instalovatelná verze aplikace
- manual.pdf – manuál k aplikaci