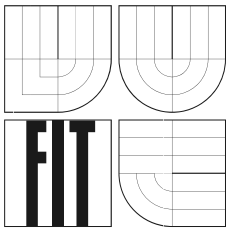
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Jabber/XMPP robot pro připomínkovač

Bakalářská práce

2007 **Petr Menšík**

Jabber/XMPP robot pro připomínkovač

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně
dne 22. leden 2007

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Marka Rychlého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Petr Menšík
22. leden 2007

Abstrakt

Cílem tohoto projektu bylo seznámení se s otevřenou sítí XMPP, jejími principy a fungováním. Tento program je určen k automatickému připomínání událostí přes síť instant messangingu. Jeho úkolem je organizovat události jednoduché, pouze pomocí jednoduché zprávy i složitější opakované děje. Cílem bylo usnadnit správu takových událostí dle možností klienta.

Klíčová slova

XMPP, Jabber, upomínky, časové pásmo, světový čas, Psi, Gajim, Exodus, JAJC, UTC, XML, stanza

Poděkování

Rád bych poděkoval svému vedoucímu Mgr. Markovi Rychlému za vstřícný a obětavý přístup i velmi pěkné zadání. Dále bych chtěl poděkovat svojí rodině za laskavý přístup a podržení ve chvílích největšího napětí. Petrovi Wechovi velice děkuji za komentáře k tomuto textu.

Abstract

The objective of this thesis was acquaint oneself with open IM network XMPP, its principles and working. This program aims to be automatic reminding machine on instant messanging network. It's work is organizing simple and repeated events.

Keywords

XMPP, Jabber, time zone, world time, Psi, Gajim, Exodus, JAJC, UTC, XML, stanza

Obsah

Kapitola 1

Úvod

Dnešní doba klade na člověka velké nároky. Moderní člověk žije v neustálem shonu a stresu, mezi desítkami dalších lidí a znečištěném ovzduší. Mimo jiné i díky počítačům a dalším moderním vymoženostem lidstva. Ačkoliv si člověk vynalézá velké množství pomůcek, aby si svoje živobytí usnadnil, má pořád velké množství starostí a povinností, které nesnesou dalšího odkladu. Protože, na rozdíl od počítače, lidská paměť netiká ani s taktem digitálních hodin, ani člověk není schopen si uložit svoje myšlenky natrvalo, stane se, že člověk na něco zapomene. Něco, co už dávno mělo být hotové. Nebo je příliš zaměstnán svojí činností, že nezareaguje včas.

1.1 Motivace

I když organizačních programů, které často slouží jako soukromá sekretářka, je velké množství, ne vždy jsou při ruce. Protože v České republice už rozvoj internetu postoupil kupředu, je dostupný na mnoha místech, ať už je to zaměstnání nebo domov. Protože člověk připojený na internet je často připojen do sítě pomocí komunikačního programu, je možné se právě pomocí něj nechat upozorňovat. A právě takové upomínky jsou námětem tohoto projektu.

Síť pro zasílání rychlých zpráv XMPP je rozšiřitelná platforma nejen pro zasílání běžných zpráv. Je otevřená, s velmi dobře popsaným protokolem a velkým množstvím podpůrných knihoven. Protokol je od počátku navržen pro další snadné rozšiřování, aniž by ohrozil fungování již existujících programů. Síť neslouží zdaleka jenom pro výměnu zpráv, ale kromě informací o stavu může přenášet data, soubory, obrázky a jiné.

1.2 Cíle projektu

V kapitole 2 přiblížím čtenáři problematiku nejdříve obecně, poté vysvětlím princip fungování sítě XMPP. Ke konci kapitoli seznámím čtenáře s několika existujícími implementacemi serverů i klientů.

Kapitola 3 přibližuje návrh vlastního programu a velmi obecně jak bude robot fungovat.

Kapitola 4 popisuje vlastní implementaci služby a konkrétní pricip jejího fungování.

Kapitola 5 shrnuje možnosti další práce na robotovi a další možná vylepšení, které by v další fázi vývoje mohl poskytovat svým uživatelům.

Kapitola 2

Teorie

2.1 Seznámení s pojmy

2.1.1 Instant Messanging

Pro komunikaci mezi lidmi slouží od dávných dob internetu systém zvaný e-mail. Je to elektronická obdoba pošty s mnoha jejími problémy a „nešvary“. Je sice několikanásobně rychlejší než jakýkoliv pošťák, který poštu roznáší, ale přesto není myšlena na komunikaci v reálném čase. Běžný email se k uživateli nedoručuje přímo, ale doručuje se do jeho schránky. Teprve z této schránky si může uživatel nový email vyzvednout a následně přečíst. I když dnes má většina programů nastavené vybírání pošty na krátkou periodu, nejčastěji 5 minut, stále to není dost rychle ani na pomalý rozhovor. Protože lidé si rádi vykládají, rádi by si vykládali trošku rychleji.

A to je právě účel programů *Instant Messangingu*, volně přeloženo programů pro rychlou výměnu zpráv. Na českém internetu se podobné programy stále častěji označují „kecálky“. Tyto programy už nemají žádnou schránku, do které nejprve něco musí doručit. Tyto programy se naopak snaží o doručení přímé, od uživatele k uživateli. Komunikace v těchto sítích se nazývá komunikací v téměř reálném čase. Slovíčko *téměř* je na místě, protože není definované doba, za kterou musí být zpráva doručena. Zasíláním zpráv v reálném čase chápeme v dnešním internetu spíše věci synchronizované s časem, kde zpozdění má nežádoucí efekt na kvalitu služby. Systémy běžící v reálném čase jsou například řídící jednotky automobilových motorů, kde zpoždění jedna vteřina v žádném případě nastat nesmí. Naproti tomu, v běžné komunikaci mezi lidmi, nehraje jedna nebo dvě vteřiny zásadní roli.

Problém těchto programů ovšem je, že, na rozdíl od emailu, nepochází ze stejného návrhu, ani nemají zveřejněný protokol popisující komunikaci v jejich síti. Dokonce provozovatelé takových sítí obvykle vůbec nestojí o to, aby se připojoval se někdo do jejich sítě s jiným programem, než je jejich vlastním. Tím pádem jsou všechny takové sítě oddělené jedna od druhé a jejich uživatelé musí být všichni na té stejné síti, pokud spolu chtějí komunikovat. Tuto mezeru vyplňuje právě protokol Jabber.

2.1.2 Jabber/XMPP

Prvním neznámým pojmem je `Jabber/XMPP`. Slovo Jabber znamená v překladu brebentit nebo breptat. Jako protokol vnikl v roce 1999 a za jeho vznikem a prvotní implementací stojí Jerremie Miller. Jednou z jeho unikátních vlastností je, mimo mnoha jiných, také jeho otevřenost. Protokol je založený na otevřeném a uznávaném standardu `XML`. Jabber přinesl otevřenost a jednoduchý popis fungování protokolu.

Narozdíl od ostatních protokolů nebyl prosazován jednou firmou se svým oficiálním klientem, ale definoval jednoznačný způsob, jak spolu mají jakékoliv dva programy komunikovat. Výběr XML v kombinaci s XML namespaces přinesl navíc možnost rozšiřovat protokol o nové prvky bez obav, že služby a programy podporující pouze staré vlastnosti přestanou fungovat. V té době byl ještě protokol označován jenom Jabber.

Přestože byl protokol Jabber ve své době stabilizován, existovala dostatečně stabilní implementace serveru a uživatelé přibývali, pořád protokol připomínal jenom jednu síť z mnoha. Aby se předešlo pochybnostem o tom, že protokol není jenom dalším z řady již existujících IM síti, jako jsou ICQ nebo MSN Messenger, byl navržen nový, neutrální, název pro protokol *Extensible Messaging and Presence Protocol* – XMPP. Až pod názvem XMPP byl protokol uznán jako doporučení IETF pod označením RFC 3920 [4] a RFC 3921 [5].

XMPP je kompatibilní protokol s původními specifikacemi Jabber protokolu, ale zavádí nové povinnosti. V podstatě tak starý protokol nahrazuje a pokud se bavíme dnes o protokolu Jabber, máme na mysli spíše jeho následníka protokol XMPP. Detaily protokolu budou popsány v dalších kapitolách.

2.1.3 IRC

Protokol IRC [2] je protokol určený především k diskuzím více lidí ve virtuálních místnostech. Přestože nemá mnoho společného se systémem XMPP, je jedním z mála otevřených standardizovaných protokolů, které se hojně používají. Na tomto protokolu jsou pěkně vidět nešvary způsobené věkem tohoto protokolu. Je sice otevřený, veřejně zdokumentovaný a počet programů jej podporujících lze na rukou těžko spočítat, ale je založený na posílání textových zpráv v textovém tvaru. Jedna zpráva začíná příkazem, jejich parametry a končí novým řádkem. Z toho plyne omezení protokolu — zprávu obsahující odřádkovaný text se na jediný pokus podařit nikdy nepodaří. Navíc má problémy s určováním kódové stránky, takže lidé na různých platformách vidí text s chybnými znaky místo diakritiky.

2.2 XML

2.2.1 Extensible Markup Language

XML [11] je jednoduchý hierarchický značkovací jazyk. Je odvozen od původního složitějšího značkovacího jazyka SGML (ISO 8879). Záměrem XML bylo vyvinou dostatečně jednoduchý jazyk pro přenos strukturovaných dat. Data jsou v textovém formátu a nejsou tedy ovlivněna nekompatibilitou mezi různými platformami. Jazyk byl zamýšlen k výměně velkých objemů dat mezi archivy textů, našel si ale cestu do všech odvětví počítačové techniky jako nesmírně univerzální formát pro výměnu hierarchických dat.

Následuje malá ukázka XML.

```
<?xml version='1.0'?>
<hlavni-tag>
  <prvni-potomek/>
  <druhy-potomek/>
  <treti-potomek ma='atribut'>
   <potomstvo>Případný další obsah není obsažen</potomstvo>
  </treti-potomek><!-- komentar -->
</hlavni-tag>
```

Elementy jazyka nazýváme tagy. Tag je hierarchický jednotka, která může mít svoje atributy, tedy parametry s hodnotou uzavřenou v jednoduchých uvozovkách. Může mít také potomky–tagy, nebo vlastní hodnotu uzavřenou mezi počátečním a koncovým tagem. Koncový tag je uvozen lomítken /.

Jazyk XML není jenom jednoduchým jazykem pro přenos prosté informace. Existuje například rozšíření `XLink`, neboli *XML Linking Language*, které definuje propojování a odkazování na jiné XML dokumenty nebo jejich části. Dále například `XPath` odkazuje na části dokumentu samého, tedy do jiného místa ve stejném souboru. Pro `XML` existuje velké množství možností a použití, ale hlavní význam má jeho existence pro strojové zpracování informací. Jednak umožní platformně nezávislou výměnu, ale mimo jiné také umožňuje transformaci jednoho `XML` souboru do jiného `XML` souboru pomocí transformačního jazyka `XSLT`. Zajímavostí je, že i definice tohoto jazyka jsou vlastně dalším XML dokumentem.

2.2.2 Práce s XML

Ačkoliv už jsem uvedl, že značkovací jazyk `XML` je jednoduchý jazyk, práce s ním má svoje zvláštnosti. I přesto, že formát dat je ryze textový, pro čtení `XML` souborů se zpravidla používá knihoven. Soubory mají na pohled jednoduchou strukturu, ale ke správnému čtení a zápisu se zpravidla používá specializovaných knihoven. Text může obsahovat znakové entity, které je před zobrazením uživateli převést. Knihovny pro zpracování XML najdeme dnes snad pro kterýkoliv používaný jazyk a platformu.

2.2.3 SAX

`SAX`, neboli *Simple Api for XML* je rozhraní pro sekvenční čtení `XML` souboru. Jeho nespornou výhodou je rychlost zpracování dokumentu. Protože dokument nenačítá celý naráz do paměti, jeho rychlost se nesnižuje ani při velmi velkých dokumentech. Jeho použití je však pro programátora o to složitější. SAX lze použít jenom ke čtení dokumentu, na zápis textu se tento způsob nehodí.Toto rozhraní není standartizováno žádnou organizací a je odvozeno od původní implementace rozhraní v Javě. Jednou z kvalitních knihoven tohoto typu je knihovna *expat*.

2.2.4 DOM

Document Object Model je naproti tomu rozhraní pro objektový přístup k dokumentům. Oproti SAX je způsob práce s dokumentem zcela odlišný, protože je třeba načíst celý dokument do paměti. Výhodou je snadná práce s dokumentem a intuitivní možnost pohybu dopředu i zpátky. Nevýhodou je, že při opravdu velkých dokumentech takový přístup klade velké nároky na systém. O to horší je potom dopad na systém, pokud se tento způsob používá ve větší míře na velkých souborech, například na serveru, kde běží množství dalších procesů s nároky na paměť.

2.2.5 Streaming XML

Při použití `XML` přes síť je problematické čtení došlých tagů, dokud ještě nejsou zcela kompletní. Jestliže Jabber/XMPP používá pro přenos veškerých zpráv XML, musí být schopen dekódovat XML ještě před uzavřením kompletního spojení. Jinak bychom mohli hodně těžko mluvit o rychlém zasílání zpráv, pokud by při každé zprávě bylo potřeba navázat a zase ukončit spojení. Navíc každé navazování a rušení spojení je zbytečná zátěž sítě i systému v případě, že zprávy jsou odesílány často. U systému pro rozhovor lidí se něco takového očekává pokaždé, když spolu začnou komunikovat. Je jisté, že zpracovat přijímaný text pomocí modelu `DOM` není možné, protože za žádných

okolností nebude k dispozici kompletní XML strom, který by bylo možné načíst. Po připojení XMPP server otevírá tag <stream>, který se uzavírá až po ohlášení od sítě nebo po chybě syntaxe XML.

2.3 Fungování XMPP/Jabber

2.3.1 XMPP Core

XMPP Core [4] je základní množinou protokolu XMPP. Tato část definuje základní prvky protokolu. I když to není přikázáno, obvyklá je architektura klient-server a spojení pomocí TCP protokolu. Jednou z největších předností je *decentralizace*. Neexistuje žádný centrální server sítě Jabber/XMPP a tedy není možné vyřadit celou síť pomocí napadení nebo selhání na jednom místě. Jak funguje decentralizace ukazuje obrázek 2.1.

Uživatelé se připojují na svůj vlastní server. Pokud chtějí komunikovat s lidmi z jiných serverů, požádají o autorizaci. Server automaticky rozpozná, že zpráva není adresovaná kontaktům lokálním, ale serveru jinému. K doručení zprávy se potom musí server spojit s cílovým serverem cílového kontaktu. XMPP Core definuje především způsob připojení autentizace klientů k serveru. Tato specifikace uvádí pouze podporu SASL[1] autentizace, avšak starší klienty a servery mohou používat i starší způsob autentizace pomocí SHA1 hashe hesla. Jednou z hlavních předností protokolu je široká podpora bezpečnostích prvků. Kromě již zmíněné bezpečné autorizace počítá protokol i s použitím šifrovaného kanálu pro výměnu dat. I když původní Jabber servery podporují i připojení na speciální TCP port chráněný pomocí SSL spojení, novější specifikace jasně favorizuje použití TLS negotiation [2] až po ustavení TCP spojení a úspěšném zpětném volání [3]. Speciální port chráněný šifrovaným kanálem mohou využít pouze klienti, pro spojení serverů mezi sebou tato možnost není podporována.

Ve specifikaci se jasně říká, že XMPP je zjednodušenou verzí XML a některé typy dat nesmí obsahovat. Mezi takové patří například XML komentáře, nebo speciální sekce CDATA obsahující binární kód. Veškerý binární obsah, jako jsou fotky, který je přenášen, musí být převeden do sedmibitové formy. K tomu se používá výhradně kódování base64.

Ukázka připojení k serveru

Připojující se posílá hlavičku bez id.

```
<?xml version='1.0'?>
<stream:stream
   xmlns:stream='http://etherx.jabber.org/streams'
   xmlns='jabber:client' to='im.pihhan.info' >
```

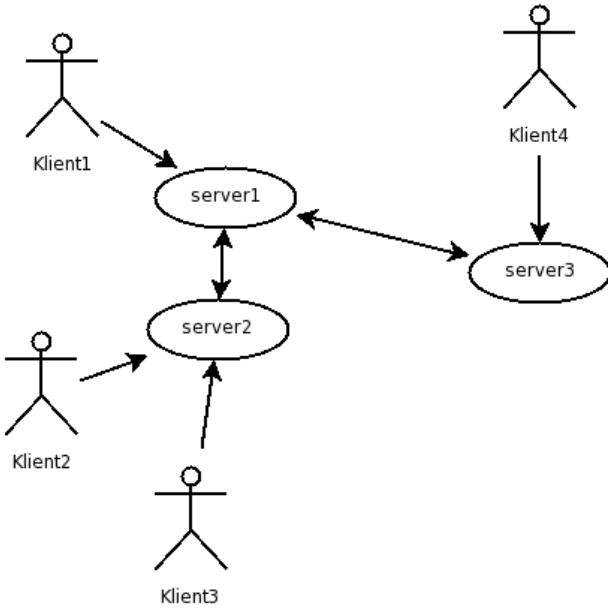
Server na ni odpoví tagem stream již s identifkátorem.

```
<?xml version='1.0'?>
<stream:stream
   xmlns:stream='http://etherx.jabber.org/streams'
   xmlns='jabber:client'
   from='im.pihhan.info'
   id='r9o2mxpyygr5h7t3nzuwl873vrvytfgko5ma3c91'>
```

[1] Simple Authentication and Security Layer

[2] Ustavení šifrovaného kanálu až po navázání nešifrovaného spojení. Server tak může použít jediný port na šifrované i nešifrované spojení.

[3] Princip zpětného volání je vysvětlen v následující kapitole



Obrázek 2.1: Ukázka decentralizace

Zpětné volání

Zpětným voláním, neboli *dialbackem*, se ověřuje totožnost vzdáleného serveru. Protože síť je decentralizovaná, neexistuje žádná centrální autorita, která by posuzovala bezpečnost a identitu serveru, se kterým se navazuje spojení. Jediná možnost je tak spolehnout se na prostředky ověření, které má k dispozici sám server.

Sice se nabízí kontrola identity serverů pomocí TLS negotiation a ověřování jejich certifikátů. Pokud však kontaktovaný server nepoužívá TLS vůbec nebo jej používá, ale jeho certifikát není ověřitelný z našeho serveru, identitu tím neověříme. Serverů které mají certifikát podepsaný sám sebou, případně nám neznámou certifikační autoritou je na serveru mnoho. Protože komunikace mezi server probíhá na celém světě, není vůbec reálné, aby všechny servery byly schopny zkontrolovat pravost každého certifikátu serveru, se kterým budou chtít komunikovat.

Navíc existují země, kde používání kryptografie je v rozporu se zákonem, a tam by nebylo možné ověřit vůbec žádnou identitu. Zkrátka omezení je příliš mnoho na to, aby se dalo mluvit o spolehlivém způsobu. Pokud se ale může připojit kdokoliv k serveru, a říct jakékoliv jméno. To je v prostředí internetu zcela nepřijatelné, protože by mohlo dojít k zaslání důvěrných zpráv nepovolaným osobám. Je tedy potřeba dostatečně jednoduchý způsob, jak ověřit pravost identity vzdáleného serveru.

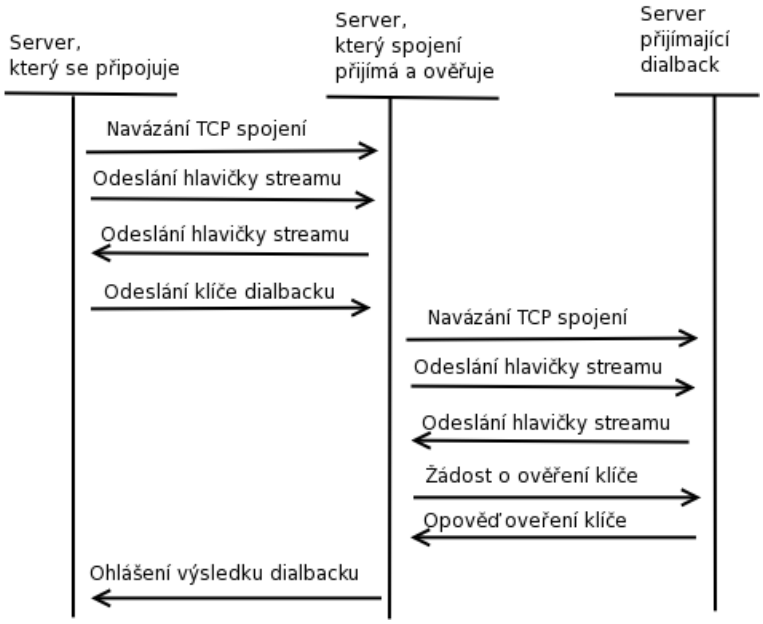
Zpětné volání funguje na pricipu sdíleného klíče. Server navazující spojení vygeneruje a pošle serveru svůj klíč pro toto spojení. Přijímající server toto spojení zatím nepotvrdí, a dokud tak neudělá, nepřenese tímto spojením žádná data. Pro přijetí klíče zahájí cílový server vlastní navazování spojení. Nejdříve musí určit, kam se má připojovat.

Pro určení cíle používá internetem ověřený systém `DNS`. *Domain Name System* je běžně používaný systém v internetu. Jeho přednostmi je vysoká dostupnost a rozšířenost a kvalitní implementace několika serverů. Hlavní úlohu v bezpečnosti dialbacku leží na bedrech systému DNS. Protože však v době masového rozšíření World Wide Webu je tato služba nepodstradatelná a podle toho jí také věnuje vysoká pozornost kvůli zabezpečení.

K získání adresy cílového serveru využívá dotaz na typ odpovědi `SRV`, a parametrem je `_xmpp-server._tcp.jmeno.serveru.net`. V odpovědi na tento dotaz může obdržet několik doménových jmen, s jejich prioritou a portem. Teprve na tato jména se připojí, podle nejvyšší priority si vybere tu nejvhodnější.

Po připojení na takto získaný server zahájí komunikaci zasláním standardního tagu `stream` s atributem `to`, který určuje jméno cílového serveru. Tím že připojující se server uvede, s jakým serverem se chce bavit, je umožněno hostování více virtuálních serverů na jediném počítači i `TCP/IP` portu.

Po zahájení komunikace server pošle sdílený klíč, který obdržel od navazujícího serveru. Pokud je `DNS` správně nastaveno, server se připojil na ten samý server, který mu předtím posílal klíč. Protože si pamatuje klíč, zkontroluje, zda takový klíč skutečně poslal. Pokud ano, odpoví kladně. Pro server, který přijímal prvotní spojení, je to potvrzení, že server je skutečně tím, kým tvrdí. Pošle tedy potvrzení o úspěšném přijetí dialbacku do původního spojení a platnost tohoto spojení potvrdí. Od této chvíle povolí odesílání i přijímaní xml paketů z tohoto spojení. Schematický princip dialbacku je lépe vidět na obrázku 2.2.



Obrázek 2.2: Princip zpětného volání

XML RPC

XMPP Core je záměrně oprostěn od konkrétních mechanismů fungování klienta pro zasílání zpráv. Standard je připraven i pro použití v jiných oblastech, kde se nebudou zasílat zprávy stejným způsobem, jako v klasické síti Jabber. Tím, že protokol je založen na XML, nabízí se mnoho dalších možností, jak využít spolehlivou základnu pro rychlé doručení částí XML dokumentů. Infrastruktura umožňuje plnou decentralizaci, přitom ale nabízí také vysokou bezpečnost. Jednou z vítaných vlastností je použití XML–RPC přes síť XMPP [1].

RPC chápeme jako *Remote Procedure Call*, tedy vzdálené volání procedur. Volání funguje tak, že vzdálené službě někde na internetu pošlu parametry a název operace, a ona mi vrátí výsledné

hodnoty zpět. V její specifikaci [12] se explicitně počítá s nasazením této technologie pomocí protokolu HTTP. Malým problémem ale je, že funkce musí vrátit nějaký výsledek v určeném čase. HTTP protokol umí jedině zadat dotaz a dostat na ni odpověď. Při čekání na ni nemůže na stejném spojení dostat odpověď jinou. HTTP servery mají omezenou dobu, po kterou podrží spojení s klientem. Pokud by služba přesáhla tuto dobu, webový server zruší spojení a celý dotaz je tak nenávratně ztracen. U `HTTP` protokolu není jiná možnost, než požádat si znovu se stejnými parametry o stejný dotaz a doufat, že tentokrát si služba uložila rozpracované dílo nebo jej stihne vypracovat rychleji.

U `XMPP` naopak žádné takové omezení není. Na jediném spojení si může služba požádat o několik požadavků naráz různým entitám, a přitom obsluhovat pouze jedno jediné spojení. Služba si pošle žádost, a dokud zůstane připojena k XMPP serveru, může čekat, jak dlouho považuje za potřebné. Server sám ji neodpojí. Tazatel se může v klidu věnovat obsluze jiných výsledků, dokud nedorazí odpověď.

2.3.2 Stanzy

Stanzy jsou jednotlivé datové tagy XML, které se mohou vyskytovat v hlavním tagu spojení `<stream>`. Často se také popisují slovem XML paket. Stanzami nejsou některé xml zprávy při navazování spojení, jako parametry `SASL` autorizace nebo žádost od navázání šifrovaného spojení TLS. Po úspěšném dokončení připojování jak klienta, tak serveru, už nesmí být odeslán žádný jiný `XML` paket, než povolená stanza. Každá stanza odesílaná mezi servery musí obsahovat příjemce v parametru `to` i odesílatele v parametru `from`. Stanza odeslaná klientem v závislosti na jejím typu nemusí mít ani jedno z nich. Potom mají takové pakety zvlášní význam pro server, který s nimi zachází speciálně. Server ale při doručování těchto stanz dalším entitám v síti tyto informace automaticky doplní.

Zpráva

První stanzou je textová zpráva. Textové zprávy pro běžného uživatele cestují v síti XMPP v tagu `<message>`

```
<message to='nekdo@example.org'>
<subject>Ukázková zpráva</subject>
<body>Zpráva pro čtenáře textu</body>
</message>
```

Zprávy mohou být 4. typů

- jednoduchá zpráva
- rozhovor
- titulek
- chybová zpráva

Jednoduchá zpráva je obdobou emailu v síti XMPP. Tato zpráva může mít volitelně i předmět ve vnořeném tagu `<subject>`. Text zprávy je obsažen v tagu `<body>`.

Informace o dostupnosti

Další stanzou je `presence`. Síť nedovoluje nikomu zaslat stav o dostupnosti bez toho, aby to ten daný člověk schválil. Pokud chcete vědět, jestli je přítel online, napřed vám to musí povolit. Pro zaslání informace o stavu dostupnosti i pro dotaz o autorizaci dostupnosti se používá tag `<presece>`.

```
<presence type='available'/>
<presence>
<show/>
</presence>
```

Informace o dostupnosti může mít několik typů.

available – stav dostupný. Ohlašuje, že uživatel je připojen a přítomen. Pokud není typ stavu uveden, předpokládá se tento.

away – stav pryč. Uživatel pravděpodobně není u svého počítače, nebo zařízení, kterým je do XMPP sítě připojen.

xa – stav `extended away`, v českých překladech často překládán jako dlouho pryč. Ohlašuje, že uživatel není a pravděpodobně nebude u počítače delší dobu.

dnd – stav nerušit. Uživatel sice u počítače je, ale nepřeje si být rušen.

unavailable – stav nedostupný. Uživatel není vůbec připojen, nebo je ve skrytém stavu.

Anglicky se autorizaci dostupnosti v systému `XMPP` říká subscription. Přesný překlad by zněl spíše „zápis“, ale protože jsou uživatelé zvyklí například z ICQ na termín „autorizace“, obvykle se používá v českých překladech termín „autorizace“.

Obecné dotazy

Pro všechny ostatní akce slouží obecné dotazy pomocí `iq` paketů. Iq packet slouží pro obecné dotazování typu dotaz a odpověď, které může při komunikaci nastat. Slouží ke zjišťování schopností klientů, k získávání verze klienta, atd.

Dotazy mají několik povinných parametrů. Prvním z nich je `id`. `Id` jednoznačně rozlišuje dotaz a odpověď od ostatních. Každý dotaz, který klient odesílá, musí mít unikátní identifikátor. Jedině takto lze jednoznačně určit, o který typ se jedná. Další povinný parametr je `namespace`. Namespace rozlišuje, o jaký typ akce se vlastně jedná. Namespace jsou definovány v jednotlivých rozšířeních [9], které registruje *XMPP Registrar*. Použití namespace umožňuje plynulý přechod na novější technologie a rozlišuje jednotlivé tagy. S jeho pomocí klienti snadno určí, zda takový dotaz zpracovat umí, nebo ne. Namespace je obsažen uvnitř `iq` tagu v jeho prvním potomkovi. Jméno tagu potomka se může lišit v závislosti na rozšíření, zpravidla jím je ale tag `query`. Posledním povinným parametrem je `type`, který určuje, o jaký úkon se tento dotaz pokouší.

get – žádost o čtení hodnoty nebo položek. Asi nejčastější typ vyvolaný klientem.

set – žádost o uložení nebo nastavení.

result – výsledek jednoho z předchozích. musí obsahovat stejné `id`, jako původní dotaz. Označuje úspěšný výsledek a v sobě nese výsledná data.

error – žádost skončila chybou. Taktéž musí obsahovat `id` dotazu, který chybu vyvolal. Obsahuje jednak hlášení s chybou a obvykle také tělo dotazu, který chybu vyvolal jako dalšího potomka.

Následuje příklad iq packetu.

```
<iq type='get' id='aab1a' >
<query xmlns='jabber:iq:roster'/>
</iq>

<iq type='result' id='aab1a' >
<query xmlns='jabber:iq:roster'>
<item subscription='from' jid='icq.gajim.org' />
<item subscription='both' name='Jan Novák' jid='649465485@icq.gajim.org' />
</query>
</iq>
```

2.3.3 XMPP IM

`XMPP IM` [5] definuje rozšířené vlastnosti specifické pro výměnu zpráv a výměnu dostupnosti. Jedna z hlavních částí je také popis práce se seznamem kontaktů – *rosteru*.

Požadavky na základní fungování instang messangingu definuje standard takto.

1. Výměna zpráv s ostatními uživateli.
2. Výměna informace o dostupnosti s ostatními uživateli.
3. Správa autorizací kontaktů
4. Správa položek v seznamu kontaktů. Systém `XMPP` nazývá seznam kontaktů pojmem Roster
5. Zablokovat komunikaci k nebo od určitých uživatelů

První tři body již byly popsány v předchozích kapitolách. Správu seznamu kontaktů zajišťuje `iq query` s namespace `jabber:x:roster`. Tento protokol zajištuje vyzvednutí seznamu po přihlášení klienta. Vyzvednutí a správa kontaktů na serveru je pro klienty povinná. Uživatel není omezen kontakty uloženými na svém počítači, protože podle specifikace každý klient musí pracovat s kontakty na serveru. Tato správa kontaktů mimo jiné umožňuje připojení několika klientů s různým `resource` na stejný účet a udržuje je všechny synchronizované. Kontakty, kterým přijme jeden z připojených klientů autorizaci, jsou automaticky serverem poslány všem zbylým jako nový kontakt do „rosteru“. Tato operace se nazývá *roster push*.

Blokování komunikace mezi určitými uživateli není v XMPP IM definováno přesně. Buď blokování komunikace zajišťují klienti ve vlastní režii, nebo ji blokuje server podle pravidel rozšíření *Privacy Lists* [3]

2.3.4 Jabber ID

Jabber ID nebo zkráceně `JID` je jednoznačný identifikátor v síti XMPP. Identifikátor má 3 části. První je uživatelská část, která určuje jednoznačně uživatele na jeho serveru. Druhá část je plné doménové jméno serveru, která jej jednoznačně určuje v síti. Třetí je pak *resource*, tedy zdroj. Zdroj rozlišuje jednotlivé připojené instance daného uživatele. Plné `JID` může vypadat například následovně.

pihhan@jabber.cz/Psi

2.3.5 XEP

XMPP Extensions [9] jsou další zdokumentovaná rozšíření XMPP IM, která přidávají další funkcionalitu nad rámec základního protokolu XMPP. Bez nich je XMPP IM pouhý systém zasílání zpráv s velmi omezenými vlastnostmi.

2.3.6 Service Discovery

Jedním z důležitých rozšíření je *XEP-0030*, Service Discovery. Toto rozšíření slouží ke zjišťování dostupných služeb a jejich podporovaných vlastností. Tuto úlohu mělo dnes již historické rozšíření Jabber Browsing (XEP–0011) a Agent Information (XEP-0094). Obě tyto rozšíření nahrazuje plně a mnohem lépe. Protože jabber má velké množství rozšíření, je často potřeba zjistit, která rozšíření služba podporuje. Toto rozšíření nejenom že umí zobrazit dostupné služby pomocí dotazu na položky – namespace 'http://jabber.org/protocol/disco#items', ale také jejich vlastnosti pomocí namespace 'http://jabber.org/protocol/disco#info'.

Discovery Info

Každá entita má svoje vlastnosti a schopnosti. Entitou rozumíme buď službu, kontakt nebo vnořený objekt nějaké služby, například místnost v diskuzní skupině. Entita je jedinečně určená svým JID, tedy uživatelskou částí, jménem serveru a zdrojem (resource), případně může být určena i pomocí node. Uživatelská část i zdroj může chybět. Každá entita podporující Service Discovery musí umět vrátit seznam podporovaných vlastností a identit. Identity rozlišují typ a zaměření entity. Výčet možných typů na rozdíl od předchozích specifikací není daný přímo v rozšíření, ale je zaregistrován u *Jabber Registrar*. Identita se skládá z kategorie a typu. Příkladem kategorie je například client, typ u něj může být například pc u klasického PC klienta, nebo phone při připojení Java aplikací z mobilního telefonu. Jedna entita může mít vždy jen jeden typ z každé kategorie. Kromě identity vrací entita při dotazu na schopnosti také seznam podporovaných vlastností v tagu <features var='namespace'>, kde *namespace* je ten stejný namespace, který se použije v iq stanze k získání hodnoty. Tímto způsobem lze možné tedy zjistit, co všechno daný kontakt podporuje ještě předtím, než odešleme žádost o provedení. To se hodí například při vytváření nabídky pro daný kontakt, protože nemá smysl uživateli nabízet tucet vlastností, při jejichž vykonání na něj stejně vyskočí chyba neimplementováno.

Discovery Items

Každé entita může mít několik dalších entit. Jejich zjištění provádí klient dotazem s namespace 'http://jabber.org/protocol/disco#info'. Entita může vrátit několik dalších entit, které mohou být znovu prohlíženy. Lze tak vytvořit stromovou strukturu s různými položkami.

2.4 Servery

2.4.1 Jabberd 1.x

Tento server je pokračováním nejstarší implementace Jabber serveru. Je napsán v jazyce C a šířen je pod licencí GNU GPL jako *open source*. Server využívá knihovnu *expat* pro parsování. Protože

je nejstajším open-source serverem, neobsahuje některé vlastnosti nových serverů. Jeho hlavní devizou je ozkoušený běh, vysoká stabilita, nízké paměťové nároky a poměrně jednoduchá konfigurace. Server nemá problémy s obsluhou několika desítek nebo i stovek uživatelů. Server může běžet na nejrůznějších unixových systémech i na Windows NT nebo novějších. Vzhledem k využití knihovny *pthread*, která na Windows 9x nefunguje, nemůže na starších Windows běžět. V dnešní době už ale málokdo vůbec takový systém má a snad nikdo by na takovém systému nechtěl provozovat jakýkoliv server, nejenom *jabberd*. Server je aktivně vyvíjen jediným vývojářem Mattiasem Whimmerem, v den psaní práce byla aktuální stabilní verze 1.6.0.

Pro jabberd 1.4 existuje celá řada služeb řada komponent, využívajících její API. Je to například implementace `mu-conference` poskytující *Multi User Chat* (JEP-0045) pro diskuze ve více lidech, jednoduchá původní databáze uživatelů `JUD` i složitější databáze Users-Agent poskytující to stejné. Většina služeb psaných pro jabberd se ale už nevyvíjí, jenom u několika z nich se pokračuje ve vývoji. Pravděpodobně to souvisí s postupným přechodem většiny serverů na modernější *ejabberd*.

Poslední verze doplnila podporu pro platné spojení XMPP protokolu. Podporuje připojení komponenty pomocí historického XEP-0114, protokol IPv6, autentizaci pomocí SASL, nově podporu TLS při navazování spojení mezi servery. Mezi nové vlastnosti patří také podpora xml:lang ve stanzách, které umožňuje komunikovat s uživatelem v jeho rodné řeči.

Jabberd jako celek obsahuje jediný binární spustitelný soubor jabberd. Server je modulární, jednotlivé schopnosti jsou implementovány do oddělených modulů. Integrovaný XML router řídí předávání stanzy mezi obsluhou klientského připojení `c2s`, správcem sezení `jsm`, komponentami připojenými přímo k serveru a samozřejmě mezi modulem pro mezi serverová spojení `s2s`. Komponenta pro ukládání uživatelských dat je připojena také přes interní router.

2.4.2 Jabberd 2

Jabberd verze 2 není verze vycházející z jabberd 1.x, jak se přímo nabízí. Tato verze je kompletním přepisem serveru jabberd do modulárnější podoby. Stejně jako jabberd 1.x je napsán v jazyce `C`. Stejně jako Jabberd 1.x je dostupný pod licencí `GNU GPL` i se zdrojovými kódy.

Na rozdíl do Jabberd jsou jednotlivé moduly odděleny úplně a jsou to nezávislé procesy. Dohromady jsou spojeny XML routerem, který vyměňuje XML stanzy mezi jednotlivými moduly. Stejně jako jabberd má moduly pro připojování klientů (`c2s`), propojování mezi servery (`s2s`), správce sezení uživatele, který ukládá veškerá uživatelova data (`sm`). Výhodou tohoto řešení je možnost restartovat jednotlivé služby. Je tak možné změnit nastavení `s2s` serveru a restartovat jej bez odpojení všech klientů, kteří jsou připojení k jinému procesu. To velmi usnadní administrativní úkony spojené s nastavováním serveru za plného provozu. Služby se připojují přímo ke XML routeru. Rozdělení služeb se hodí při vysokém vytížení, kdy každá služba může běžet odděleně na jiném stroji.

Jabberd 2 nemá kvůli značným výkonostním problémům ukládání do XML souborů vůbec. Místo toho se spoléhá na databázi Mysql, případně Berkeley DB. Autentizačních modulů je na výběr více, například Mysql, LDAP, nebo opět Berkeley DB. Server podporuje `STARTTLS` jak pro klienta i serverové spojení, stejně tak pro připojování komponent. Podporuje spojení i protokolem IPv6, a to jak klentů, serverů i vlastních komponent. Implementováno je také Privacy Lists (XEP-0016) [3], které umožní zcela ignorovat dotěrné uživatele, kteří vás obtěžují.

Jabberd 2 nemá tolik dostupných služeb napsaných pro vlastní rozhraní. Projekt není tak ustálený a vyzrálý jako první verze jabberd. Služeb napsaných pouze pro jabberd 2 lze nalézt velmi málo. Pro zprovoznění služeb je potřeba nainstalovat JCR API, které emuluje rozhraní Jabberd 1.4, a umožní tak zprovoznit služby napsané pro Jabberd 1.4.

2.4.3 eJabberd

Ejabberd je momentálně nejintenzivněji vyvíjeným open source serverem. Server je napsaný v jazyce Erlang vyvinutým firmou Siemens pro tvorbu real-time aplikací. Že jde o jazyk velmi kvalitní napovídá to, že ejabberd je momentálně asi nejlepším serverem na vysokou zátěž. Pokud je potřeba obsluhovat víc než tisíc uživatelů, ejabberd mnohem méně zatěžuje CPU serveru než jabberd 1.4 nebo jabberd 2. V poslední době jsou servery jabberd často nahrazovány serverem ejabberd, protože z dostupných serverů umí nejvíce rozšíření.

Podporuje přihlašování pomocí `SASL`, zapezpečené připojení pomocí `STARTTLS` pro klienty i servery. Implementuje jako jediný sdílené skupiny v seznamech uživatelů, Stream Commpresion pro snížení datového toku. Server má taky několik vestavěných komponent, jako Publish Subscribe, Multiuser Chat, IRC gateway a jiné.

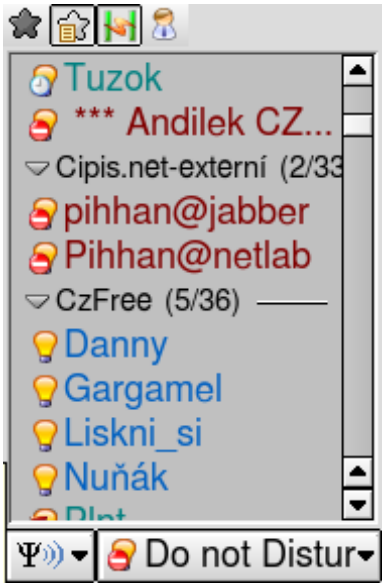
Jednou z nevýhod ejabberd je vysoká náročnost na paměť. Sice ušetří značné zatížení procesoru, paměti ale potřebuje více než oba předešlé.

2.5 Klienti

2.5.1 Psi

Psi [7] je multiplatformní Jabber klient napsaný v toolkitu Qt v jazyce C++. Tento klient je jedním z nejstarších, nejpoužívanějších a nejstabilnějších klientů pro Jabber vůbec. I když Psi není zrovna technologickou bombou podporující skoro každé rozšíření, které Jabber Working Group vydá, má poměrně bohaté schopnosti. Samozřejmou je podpora Service Discovery, která je v současném klientu nutností. Psi podporuje vyhledávání uživatelů pomocí Data Forms (*XEP-004*), přenos souborů pomocí SOCKS5 Bytestreams (*XEP-0065*). Klient také možňuje připojení více účtů zároveň, ale v poslední stabilní verzi neumí ani *Ad-Hoc commands* ani Multi-User Chat. Obojí je ale součástí neoficiálních verzí s některými vylepšenými vlastnostmi.

Přednostmi Psi je především příjemné jednoduché uživatelské rozhraní, stabilita a dostupnost na všech hlavních platformách.



Obrázek 2.3: Ukázka rosteru klienta Psi

2.5.2 Gajim

Gajim [6] je poměrně mladý XMPP klient napsaný v pythonu. Přesto podporuje asi nejvíce rozšíření ze mě známých klientů. Jako předchozí Psi je volně dostupný i se zdrojovými kódy pod licencí *GNU GPL*. Z celkem schopnostmi nabitého klienta, který měl však hodně nepříjemné ovládání, dospěl v posledním roce ke špičcce mezi Jabber klienty. Za posledních několik měsíců se z nerudného podivného klienta stal nejlepší klient pro návštěvu groupchatu, s nejlépe provedenou správou záložek a zprávách v tabech. Je dané jeho jazykem, že paměťově je trochu náročnější než Psi, přece jenom běží v interpretovaném jazyku. Umí přenosy souborů pomocí SOCKS5 Bytestreams (`XEP-0065`), Ad-hoc Commands (*XEP-0050*). Podporuje zabezpečené přihlašování pomocí SASL i STARTTLS a jako jediný z velkých stolních klientů umí připojení pomocí IPv6. Má poměrně dobré zpracování Service Discovery, ale neumí například vyhledávání v seznamech uživatelů.

Funguje na Linuxu i Windows. V na českých verzích Windows XP nefungovaly starší verze kvůli chybě při překladu proměnné s českými znaky do UTF-8. V posledním vydání je toto už opraveno a funguje spolehlivě i na českých Windows. Stejně jako Psi zvládá připojení k více účtům současně.

2.5.3 JAJC

Just Another Jabber Client není zase tak obyčejný klient, jak napovídá jeho název. Program je Jabber klient pouze do MS Windows. Na rozdíl od předešlých klientů není open source, ale je uvolněn jenom jako freeware. Jeho autorem je Mikem Ivanov, programátor, který v době nejhektičtějšího vývoje vydával nové verze každých 14 dnů. JAJC už se nějakou dobu nevyvíjí, přesto podporuje řadu rozšíření. Podporuje Service Discovery, prohlížeč služeb je dobře zpracovaný. Další hezky provedenou věcí je vyhledávání pomocí *Data Forms* (XEP-0004). Umožňuje poslat zprávu kterémukoliv kontaktu z prohlížeče služeb, což se jistě může hodit.

2.5.4 Exodus

Tento klient je jeden z nejstarších. Za jeho aktivním vývojem stojí již Peter Millard. Je napsaný v Delphi a vývoj opadává už delší dobu, stejně jako samotný jazyk Delphi. Pořád je ale na špici v počtu podporovaných rozšíření. Podporuje jak Ad-hoc Commands, přenos souborů, vyhledávání pomocí Data Forms, Service Discovery taktéž. Jako jediný z uvedených klientů jeho uživatelské rozhraní umožňuje poslat zprávu i kontaktům vyhledaným v databázi kontaktů za pomoci Jabber Search.

2.6 Rozdíly služby a klienta

2.6.1 Klient

Běžný uživatelský program pro připojení do sítě XMPP — klient — se do sítě připojuje pomocí běžného klientského protokolu. Při připojení jako klient se musí klient nejdříve zaregistrovat. Registrace na serveru probíhá pomocí rozšíření *XEP-0077: In-Band Registration*. Ne všechny servery musí mít tuto možnost povolenou, některé například mohou používat raději webový formulář. Zvláště pokud jsou to servery komunitního charakteru, mohou chtít omezit registrace jenom pro své členy. U veřejných serverů je obvyklé, že registrace je povolená. Jediné, co je potřeba k připojení je klientstký program.

Po každém připojení klienta se *vždy* musí stáhnout roster s aktuálními kontakty. Servery nedoručí žádné stanzy `<presence>`, dokud se uživatel sám nepřihlásí online. Teprve po přihlášení jsou vidět stavy autorizovaných uživatelů.

Tyhle všechny věci pro robota příliš důležité nejsou. Robot sice může uchovávat informaci o dostupnosti uživatele, ale není to zcela nezbytné. Robot se stejně tak může spolehnou na doručovací schopnosti serveru uživatele, a poslat prostě upomínku na Jabber ID bez uvedení zdroje. Pokud je uživatel online, bude zpráva doručena serverem zdroji s nejvyšší prioritou nebo poslednímu připojenému zdroji. Některé servery mohou zvolit zkopírování zprávy všem připojeným zdrojům, standard tohle chování nedefinuje.

Služby serveru

Klasický klient má mnoho služeb zprostředkovaných serverem. Například při přijetí nebo odmítnutí autorizace server automaticky vyvolá „roster push“ a přidá nový kontakt automaticky do rosteru klienta, pokud to už nestihl udělat klient sám. To sice usnadňuje práci klientovi, ale pro robota nestarajícího se o roster ani stavy není toto podstatné. Těchto služeb většina robotů nevyužívá. Robot se orientuje spíše na vstup ze zpráv, případně `iq` paketů a příkazů.

Robotů v dnešní síti XMPP existuje celá řada. Velice často je ale problém tyto roboty najít. V běžné kartotéce uživatelů roboti nebývají. Celá řada užitečných robotů je ztracena někde na diskuzních fórech a na zapadlých webech, kde je každodenní uživatelé jenom náhodou najdou. Snižuje se potom jejich přínos síti, protože většina uživatelů robota nevyužívá z pouhé nevědomosti. I když kontakt je možné vložit do hlavního seznamu služeb, snad žádný z robotů není uveden v seznamu služeb na svém serveru.

Omezení klienta

Pokud je robot v režimu klienta, množství klientů neumí poslat jemu a jen jemu jiný stav než globální. Například pokud robot má za úkol reagovat pouze v případě, že cílový uživatel je online. Uživatel může chtít odložit událost jednoduchým zasláním stavu offline robotovi. Robot si pak bude myslet, že uživatel je odpojený a upomínky si odloží. Až se uživatel rozhodne, že už si od robota přeje přijímat události, pošle mu zase znovu zasláním stavu, tentokrát stav online – *available*. Tento jednoduchý princip má však svoje úskalí. Většina klientů takovou operaci nepodporuje. Buď se opírají o použití Privacy Lists, které je ovšem v tomto případě trošku nepraktické. Pokud se odložení doručení týká několika mála minut, je to zbytečně složitá operace.Někteří klienti neumí ani jednu z možností. Pro běžné kontakty umí odeslání zvláštního stavu jenom program JAJC.

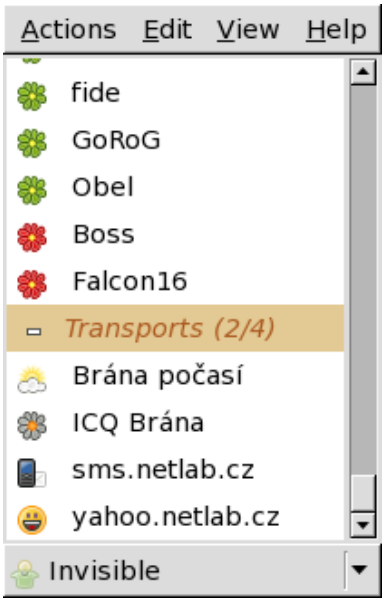
Mnoho speciálních funkcí na klienty nefunguje. Klienti například nepředpokládají, že někdo bude chtít vyhledávat na kontaktu s vyplněnou uživatelskou částí.

2.6.2 Služba

Služba v systému XMPP je Jabber ID bez udání uživatelské části. Obvykle je taková služba samotný server, adresář uživatelů, konference více uživatelů nebo brána do cizí sítě. Službě neposkytuje server mnoho výsad, přesněji řečeno server jí poskytuje pouhé připojení, jinak vůbec nic. Služba si všechno ostatní musí řídit sama. To zvyšuje složitost, na druhou stranu to službě dává možnost se chovat jinak, než je obvyklé pro obyčejného klienta. Pokud služba chce mít uživatelské jméno ve formátu běžného uživatele, tedy Jabber ID s uživatelskou částí, serverem a případně i zdrojem, může nabízet takový kontakt svým uživatelům. Formát neodporující formátu JID je zcela v režii služby. Může například použít zdroje k odlišení úkolu, k výběru údaje nebo podobné věci. Může tak stejnou zprávu obsluhovat trochu jinak, podle toho, jaký zdroj byl cílovým zdrojem.

Služba se obvykle do rosteru uživatele přidává pomocí In-Band Registration. Při registraci například k ICQ bráně si zaregistrujeme u služby jméno a heslo do ICQ sítě. Po úspěšné registraci požádá entita o autorizaci pro kontakt icq brány, tedy jméno brány bez uživatelské části. Někteří klienti takový dotaz s ohledem na předchozí uživatelovu žádost o registraci automaticky povolí (Exodus), jiné si i tento dotaz nechají od uživatele potvrdit (Gajim). Faktem ovšem je, že tento kontakt je ve všech klientech speciálně. Prakticky všechny programy umožňují se přihlásit a odhlásit od této služby. Ve své podstatě to není nic jiného, než zaslání vlastního stavu cílené pouze určené komponentě, nikoliv rozeslání všem uživatelům. Přesně to stejné, co jsem popsal o pár odstavců výše funguje v úplně každém klientovi, ve kterém jsem to zkusil. Výjimkou budou snad jenom multiprotokolové klienty, které berou Jabber jenom jako další síť s běžnými kontakty. Takové programy zpravidla nijak nerozlišují typ služeb. Většinou ani neumí registraci nebo vyhledávání kontaktu, takže práce se službou je pro ně velká neznámá.

Služby jsou také dost často v klientovi zobrazované jinak, nebo alespoň setříděny do skutečné nebo virtuální skupiny v klientovi. Většina klientů tyto služby seskupuje na spodní straně rosteru ve virtuální skupině transporty nebo podobně. Ukázka zobrazení virtuální skupiny *Transports* je na obrázku 2.4.



Obrázek 2.4: Ukázka seskupení transportů

U serveru Jabberd 2 se služby po úspěšném připojení ke XML routeru automaticky zaregistrují do seznamu služeb serveru. Ihned po připojení je služba nabídnutá všem uživatelům serveru a ti mají větší šanci se dovědět o přítomnosti služby velmi snadno.

Kapitola 3

Návrh

3.1 Databáze

S použitím *Unified modelling language* si přiblížíme požadavky na funkce robota. Robot bude používat pro ukládání upomínek relační databázi. Pro největší rozšířenost a dostupnost vhodné knihovny jsem zvolil databázi `MySQL`. Robot neklade na schopnosti databáze žádné složité požadavky, většina operací je obyčejné vložení, úprava a čtení s minimem parametrů.

3.1.1 Případy použití

Obrázek 3.1 zachycuje hlavní případy použití robota. Prvotní uživatelovou činností před počátkem interakce bude registrace ke službě. Registrace může probíhat jenom jako přidání kontaktu a potvrzení autorizace v případě klienta, nebo v případě komponenty jako klasická registrace ke službě. Další činnosti vyplývají ze zadání úkolu. Aby měl celý robot smysl, musí být možné vložit novou upomínku. Tuto upomínku bude také možné změnit nebo předčasně zrušit. Aby si uživatel mohl něco změnit, bude napřed potřebovat vypsat již existující upomínky.

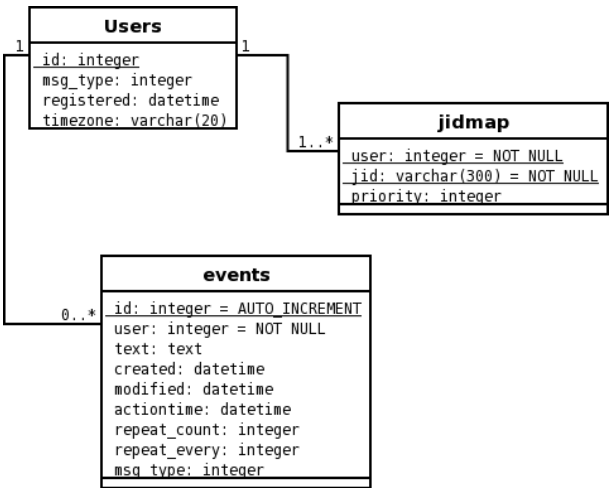
Obrázek 3.1: Případy použití robota

3.1.2 ER diagram

Při registraci se uživateli vytvoří unikátní číslo, které pro něj bude používáno interně. Pro jednoznačnou identifikaci by sice stačilo Jabber ID uživatele, ale použití jedinečného identifikátoru navíc

umožní spojit několik uživatelů do jediného. Někteří uživatelé mají například více účtů na různých serverech, které používají jenom v některých případech. Příkladem může být oddělený účet s minimem kontaktů určeným pro použití na mobilním telefonu, kde velké množství kontaktů může snadno způsobit nedostatek paměti a pád aplikace. Na takovém účtu uživatel má jenom kontakty pro něj důležité. Další možností je obdržení dalšího účtu v XMPP síti například práci nebo škole, který je aktivní ne vždycky. Jinou alternativou je využití dalšího kontaktu jako alternativního zúůsobu pro upozornění, pokud nejsem aktivní. V takovém případě může s výhodou posloužit například SMS brána, která doručí upomínku až na mobilní telefon, pokud není uživatel dostupný přímo přes XMPP síť.

Každý uživatel bude mít ještě několik položek pro drobné uzpůsobení robota. Zásadní položkou je pouze ruční nastavení časového pásma. Nejvíce položek bude mít tabulka `events`, která bude obsahovat parametry jednotlivých upomínek s určením jeho vlastníka. Pro větší názornost poslouží obrázek 3.2. U tabulky `jidmap` je primárním klíčem dohromady jak číslo uživatele, tak Jabber ID k němu přiřazené. To, že dohromady tvoří primární klíč, zaručuje unikátnost propojení. Není tedy potřeba žádný další index na omezení unikátnosti.



Obrázek 3.2: ER Diagram databáze

3.2 Třídy robota

Jako jazyk pro implementaci jsem si vybral jazyk C++, i když pro podobné roboty se nejčastěji používá nějaký skriptovací jazyk. V případě sítě XMPP je to nejčastěji Python. Při výběru je však důležitá úroveň zkušeností programátora s daným jazykem. I když já mám největší zkušenosti s jazykem C, tento jazyk se mi pro zpracování robota nezdál vhodný. XMPP je založené na XML a pro jeho zpracování jako hierarchické struktury se nabízejí objekty. Protože trochu zkušeností mám i s C++, i když minimum s objektovým programováním, a přece jen je C++ příbuzný s jazykem C. Nezkušenost s objektovým programováním ztěžuje vytvoření slušného návrhu, protože objekty je nutné navrhnou předem.

3.2.1 Knihovna pro XMPP

Jako knihovnu pro práci s Jabberem jsem si vybral knihovnu Gloox. Sice ani s tou jsem neměl žádné zkušenosti, ale je jednou z mála dostupných knihoven pro jazyk C++ a programové rozhraní má kvalitně zdokumentované.

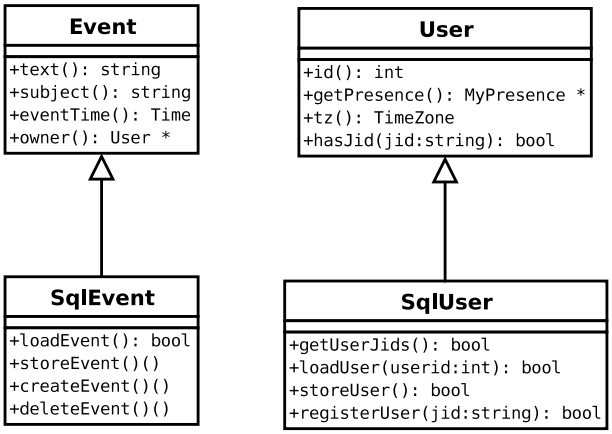
3.2.2 Knihovna pro MySQL

Protože už padlo rozhodnutí pro jazyk C++, bylo by hloupé použít jazyka C pro přístup k databázi. Zvolil jsem tedy knihovnu MySQL++, která využívá výhod přetypovávání a šablon v jazyce C++.

3.2.3 Třídy Event a User

Jako abstrakci tabulek z databáze jsem vytvořil dvě hlavní třídy, které mají uchovávat informace o uživateli i jeho nastavení a událostech uživatele. Třída User obsahuje údaje o uživateli a jeho nastavení, jeho seznam Jabber ID a obdržené informace o dostupnosti uživatele. Dostupnost uživatele — presence — se do databáze neukládá, nemělo by to smysl.

Pro práci s databází jsem odvodil z obecných tříd pro práci s upomínkou a uživatelem jejich databázové ekvivalenty, které navíc k jejich funkcím přidají metody pro čtení a uložení do databáze. Vztah těchto dvou tříd zachycuje obrázek 3.3. Základní třída User zpracovává dostupnost a online stav uživatele. Nese také nastavení pro tohoto uživatele, schopnosti jeho klienta a jeho časové pásmo. Třída Event bude obsahovat všechny operace pro zobrazení upomínky, práci a nastavování jejích parametrů, zpracování formuláře vyplněného i vyplnění formuláře úprav z parametrů upomínky. Třída SqlUser bude pouze spravovat uložení nastavení uživatele, načtení při přihlášení uživatele a registraci nového. SqlEvent podobně bude spravovat pouze načtení, uložení a vytvoření nové upomínky. Převod na rodičovskou třídu se zajistí přetížením operátoru přiřazení.



Obrázek 3.3: Dědění tříd Event a User

3.2.4 Třída Time

Třída Time je abstrakcí pro práci s časem. Její výchozí konstruktor naplní strukturu aktuálním časem. Třída implementuje nastavování času, posun času vpřed i zpět. Mimo jiné porovnává a dekóduje data různých formátů.

Její význam je klíčový, protože bude zpracovávat posunutí času, konvertovat formát data z různých zdrojů, například mezi XMPP rozšířením a SQL databází. Tato třída bude také provádět přepočítávání času do různých časových pásem.

3.2.5 Třídy Lexical a Syntax

Pro zpracování příkazů od uživatele slouží třídy Lexical a Syntax. Třída Lexical postupně čte jednotlivá slova z příkazu a odesílá je formou třídy Token do třídy Syntax. Pomocí konečného automatu třídy Syntax buď nahlásí chybu nebo připraví upomínku pro uvedenou operaci.

3.2.6 Princip

Robot bude využívat neblokující operace na socket připojení k Jabber serveru. Periodicky bude kontrolovat, zda některé upomínce nevypršela doba čekání a zda se tedy nemá odeslat. Pokud jsou takové upomínky nalezeny, nahrají se pomocí metod tříd a provede se odeslání upomínky. Pokud upomínka byla nastavená s periodou opakování,změní se datum aktivace upomínky na příští čas. Pokud je uvedeno omezení počtu opakování, počet opakování se sníží. Poté se uloží zpět do databáze. Pokud byla upomínka jednorázová nebo počet opakování už je roven nule, upomínka se vymaže z databáze.

Ačkoliv tento způsob bude vyžadovat časté zásahy do databáze, má i svoje výhody. Můj původní záměr byl načíst datum upomínky, která jako první vyprší. Stanovit rozdíl mezi tímto datem a aktuálním časem, a po tuto dobu potom jenom periodicky obsluhovat spojení do sítě Jabber. Potom jsem usoudil, že ve spojení s indexem na tabulku času nebude periodické čtení vypršených upomínek nějakou znatelnou zátěží. Navíc při periodickém čtení s krátkým intervalem může upomínkovač spolupracovat třeba s webovým formulářem, přes který bude moci uživatel přidat další podmínku pohodlně do databáze, a na Jabber bude upozorněn, až vyprší.

3.2.7 Posun času

Robot se také bude snažit automaticky kompenzovat posun času v různých časových pásmech. Pro automatické nastavení se použije jednoho ze způsobů pro zjišťování času pomocí rozšíření *Entity Time* (XEP-90) nebo *Entity Time* (XEP-202). Většinou asi bude implementován pouze první způsob, protože druhý, i když pro zpracování výhodnější, je zatím v experimentálním stádiu.

Kapitola 4

Implementace

4.1 Časové zóny

Protože na světě je různý čas a ne každý uživatel má čas stejný, jeden robot může mít špatné údaje o uživateli. Robot sám je v jedné části země, kde čas se nemění. Ale jeho uživatelé mohou být v sousedních pásmech, nebo i na jiném světadílu. Aby mohli uživatelé smysluplně zadávat svoje upomínky nejen v relativní formě (upozorni mě za 15 minut), ale i se zadáním přesného času, musí robot vědět, jaký čas uživatel má.

Když uživatel zadává čas, obvykle chce zadávat svůj čas, který je v místě jeho pobytu. Robot však musí nejprve zjistit, jaký čas uživatel má, protože ne nutně musí být ve stejném pásmu a mít tak čas stejný. Uživatel by byl jistě nespokojen, pokud by si zadal čas a jeho upomínka, která měla přijít za hodinu, přišla za hodin šest. Robot tedy bude pracovat interně s časem *Universal Coordinated Time*, který je univerzálně použitelný po celém světě a uznávaný.

Časy od lokálních uživatelů si bude robot při každé úpravě převádět do univerzálního času, aby mohl zároveň porovnávat čas od uživatelů v různých časových zónách. Při každém dotazu uživatele převede jeho čas zpět na jeho místní čas.

4.1.1 Zjišťování času

Zjištování času uživatele závisí na schopnostech jeho klienta. Pokud klient umí alepoň nějakou metodu určení času, není třeba obtěžovat uživatele. Je maximálně vhodné, aby uživatel nebyl nucen nastavovat si svůj čas ručně, pokud to jde jinak.

Čas klienta se zjišťuje při každém přechodu uživatele do stavu online. U každého uživatele si zjistí jeho vlastnosti pomocí *Service Discovery*, a zkusí zjistit čas. Správně by měl zkoušet čas jenom u klientů, kteří v sekci `features` odpovědi *Service Discovery*. Někteří klienti ale nemají vypsány všechny podporované vlastnosti a není možné se na tyto informace na 100 % spolehnout. Protože protokol má jasně definovanou odpověď při nepodporovanén rozšíření, dotaz se pošle bez ohledu na ohlášenou podporu. Všechny zkoušené klienty umí korektně nahlásit, pokud protokol neumějí. Pokud by ani neuměly správně ohlásit, že rozšíření neumí, robot bude měnit časové pásmo jenom v případě, že dostane platnou odpověď. Pokud dostane chybové hlášení nebo vůbec nic, předpokládá původní nastavení.

Protože uživatel může zkoušet různé klienty a některé rozšíření umí, jiné nikoliv, bude si robot zaznamenávat poslední podporované časové pásmo. Jakmile jednou robot obdrží určení časového pásma, bude předpokládat, že tento kontakt moc necestuje a časové pásmo má pořád stejné. Pokud cestovat bude, pro správné fungování korekcí musí jeho klient podporovat určení časového pásma.

4.1.2 Zkratky časových pásem

Když jsem plánoval, jak robot bude fungovat, předpokládal jsem podporu alespoň historického rozšíření ve všech dnešních klientech. Rozšíření XEP–90 je definováno již hodně dlouho, a používalo se ještě dlouho před tím, než vůbec bylo zdokumentováno jako rozšíření s číslem 90. Toto rozšíření umožňuje získat od vzdálené entity jeho čas, zkratku jeho časového pásma a čas zobrazený ve formátu běžném pro tohoto uživatele. Poslední část nemá valné uplatnění, protože není možné ji nějak strojově zpracovat. Pro robota tedy nemá absolutně žádný význam. Ani čas klienta není příliš zajímavý, protože je udáván v mezinárodním čase UTC a tedy by měl být stejný jako čas robota. Je možné sdělit uživateli rozdíl mezi časem robota a uživatele, ale pro tyto prostředky jsou mnohem pokročilejší protokoly.

Nejdůležitější je parametr se zkratkou časové zóny. Jedině tento je schopen poskytnout robotovi užitečnou informaci, a tedy v jakém časové pásmo je. Co jsem ovšem vůbec netušil v době teoretického návrhu robota bylo, že tyto zkratky jsou poměrně chaoticky rozděleny po světě. Neexistuje žádná specifikace podobná seznamu zkratek národů nebo zkratek států. Najít posunutí času ze zkratky není tak jednoduché, jak jsem si původně myslel.

Mým záměrem bylo využít volání knihovny, kterému zadám čas a zkratku, a dostanu čas v této zóně. Nebyl jsem zase tak překvapený, když jsem takové volání nenašel ani na `POSIX`u, ani ve `Win API`. Co mě ale překvapilo více, že jsem nenašel ani žádné systémové volání pro zjištění posunutí času v jiné zóně. Vlastně práce s jiným než mým vlastním časovým pásmem je docela problematická, v jazyce C ani C++ není dostupné nějaké jednoduché a přenositelné řešení. Takové problémy měli pravděpodobně i jiní vývojáři nejen Jabber klientů a proto rozšíření využívající zkratku časové zóny je v novém experimentálním rozšíření nahrazeno mnohem praktičtějším posunutím uvedeným číselně v hodinách a minutách.

Zvolil jsem tedy vytvoření tabulky vlastní, vytvořenou ze seznamu časových pásem ze serveru Time Genie [8]. I když tento seznam není oficiální, byl nejúplnějším seznamem časových pásem. Sice není zcela nutné mít seznam časových pásem úplně pro celý svět, zvláště když stávající verze komunikuje pouze česky. Pravděpodobnost, že by tolik česky mluvících lidí cestovalo po světě není velká. I když autor většiny rozšíření i specifikace XMPP Peter Saint-Andre česky trochu umí a v některých rozšířeních češtinu můžeme zahlédnout. Převody pomocí zkratek se tedy provádí vlastními funkcemi, které vrací posunutí v minutách. Existují země, například střed Austrálie, kde posunutí není po hodinách, takže bylo nutné mít menší jednotku.

4.1.3 Přechody letního a zimního času

Původně jsem měl si myslel, že posunování letního a zimního času mi pomůže nastavovat systém. Protože jsem ale zjistil, že kromě změny vlastního časového pásma mi systém žádnou pomoc nenabízí, hledal jsem další řešení.

Posunování mezi letním a zimním časem se zavedlo někdy ke konci 18. století z důvodu úspory elektrické energie. Protože se v letních měsících posouvá doba svítání a západu slunce na pozdější hodiny, je nutné délší dobu svítit. Aby se nemuselo svítit tak dlouho a den začínal stejně dlouho po rozednění, posouvá se v létě čas o hodinu zpět. Takové posunutí času některé státy aplikují, jiné ne. Neplatí tedy, že v jedné časové zoně se posun času používá a v jiné ne. Sice většina zkratek časových zón obsahuje písmeno `S` od anglického slova *Summer* — letní, ale určit, zda někdo používá nebo nepoužívá posunutý letní čas z pohledu robota je téměř loterie.

Tady je například užitečné, když klient vrací svoje časové pásmo jako název nebo zkratku. Pokud tak učiní, můžu u mě známých zón jasně říct, že u této časové zóny používají letní čas. Například pro středoevropskou zónu `CET` se používá pro označení letního času `CEST`. Z pohledu

robota tak můžu tipovat, zda tento uživatel bude mít od nějakého data čas jiný, i když zůstane na stejném místě.

Jeden z problémů ale spočívá také v tom, že datum přechodu z a na letní čas není vždy jasně dané. Ještě v roce 1982 se čas v České Republice měnil na letní o půlnoci (viz [10]), od té doby se mění ve dvě hodiny v noci. Ještě donedávna vyhlašovaly data posunu času české úřady samostatně, po vstupu do evropské unie je posun času synchronizován ve všech státech a čas posunu je závazný. Tyto časy nejsou nijak vhodně plánovány pro výpočetní techniku, ale spíše s ohledem na pracující národ a praktické problémy života. Změna tak probíhá poslední březnovou neděli v noci a vrací se v poslední říjnovou neděli zpět. Už jenom to, že posun není konkrétní datum, ale každý rok vychází jinak, je docela problematický. Určit pro konkrétní datum a čas, jestli v uvedené době oproti momentálnímu času se posune nebo ne je složité a závisí na znalosti zvyklostí v dané zemi.

Robot nemá absolutně žádnou představu o tom, v jaké zemi se pohybuju. Jediné vodítko může být časová zóna, případně jazyk uživatele. V anglicky mluvících zemích, které jsou snad na všech kontinentech však jazyk neurčuje vůbec nic.

4.1.4 Řešení posunu času

Kvůli právě vyjmenovaným důvodům považuji korekci letního času s ohledem na jiné státy než Českou Republiku téměř za neproveditelnou. Pokud tedy uživatel zadá začátkem března, že chce být upozorněn za měsíc v 15 hodin, robot si jeho čas převede do UTC a uloží. Když ale bude upomínku odesílat, uživatel už změnil čas a 15 hodin nebude jeho aktuální čas. Upomínka tak dojde o hodinu špatně.

Moje řešení tohoto problému spočívá v periodickém testování uživatelovy časové zóny. Pokud má uživatel podporu v klientovi, při každém připojení se mu zkontroluje čas. Pokud robot zjistí, že se časové pásmo uživatele neshoduje s tím posledně uloženým, provede korekci všech jeho uložených upomínek a upraví čas jejich aktivace. Tímto způsobem se dodrží doba doručení, i když se mezitím změní časové pásmo. Lze tak opravit upomínky i o více hodin, než o jedinou při posunu na letní čas. Například pokud poletím za oceán na dovolenou, bude čas najednou alespoň o 6 hodin posunutý. Robot po přihlášení zjistí, že časová zóna se změnila, a vypočítá rozdíl časů. Poté opraví všechny upomínky a ty, které už měly být doručeny dodatečně zašle.

Tento způsob má také několik nedostatků. Není jisté, zda když jsem si zadal před změnou nějaký čas, nemyslel jsem tím odpovídající čas bez ohledu na to, v jaké zemi budu nebo jestli se změní čas na letní. Robot provede korekci vždy, aby byla dodržena určená hodina. U každé upomínky si tak bude ukládat ještě časové posunutí, při kterém byla vytvořena. Protože ale většinou opomínka s uvedenou hodinou míněna jako přesný čas, předpokládám že v převážné většině případů bude zvolené chování nejvhodnější.

4.1.5 Podpora klientů

Tohle chování ovšem nebude fungovat, pokud pravidelně používaný klient neumí sdělit robotovi svoje časové pásmo. Bohužel jsem zjistil, že ani v nejnovější verzi klienti Psi a tkabber neumí sdělit svůj čas. Psi je jeden z nejlepších a nejpoužívanějších klientů, bohužel neumí ani jednu z možnosti. Gajim byl jediný, který podporoval i novější experimentální rozšíření s uvedením časové zóny pomocí číselného posunu, nikoliv zkratky. Bez klienta s podporou protokolu je posunování času plně v rukou uživatele.

4.2 Správa upomínek

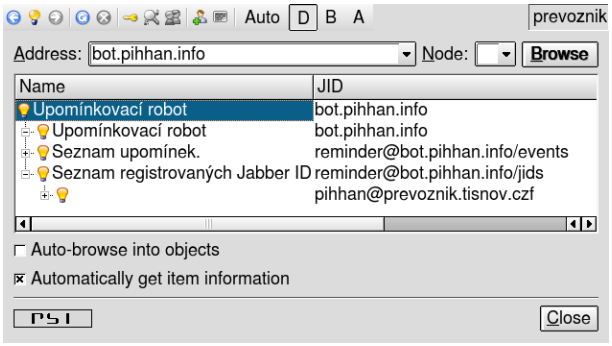
4.2.1 Registrace

Před započetím používání robota se uživatel musí zaregistrovat. K registraci nepotřebuje ani heslo, ani něco jiného kromě vlastního Jabber ID. Registrace je nutná k vytvoření čísla uživatele, které je pak využíváno při prohledávání tabulek v databázi. Registrace se provede použitím klasické intergrované registrace pomocí prohlížeče služeb. Případně stačí si přidat kontakt na robota a požádat jej o autorizaci. Robot sám autorizaci vydá a pokud ještě daného uživatele nemá zaregistrovaného, potom si jej zaregistruje.

Protože je robot služba, uživatel si může přidat jakýkoliv kontakt spadající pod jméno služby. Například jméno serveru — služby je `bot.example.org`, potom si můžu přidat jakýkoliv kontakt s různým identifikátorem uživatele a částí serveru shodnout se jménem služby. Například `reminder@bot.example.org` je ve své funkčnosti ekvivalentní kontaktu `kremilek@bot.example.org`. Uživatel si může zvolit podle svojí fantazie. Robotovi je celkem jedno, přes který kontakt přijímá komunikaci. Svoje odpovědi bude posílat vždy ze zdroje, na kterou přišla žádost, takže uživatele to nijak neovlivní.

4.2.2 Prohlížení služeb

Prakticky všechny moderní klienty specializované na jabber mají prohlížeč služeb s využíváním *Service Discovery*. Způsoby zobrazení se liší jeden od druhého. Na obrázku 4.1 je ukázka hlavního stromu robota s nabídnutými službami. Pomocí kombinace prohlížeče služeb a registrací je možné si pohodlně upravovat a přidávat nové upomínky. K úpravě se používá *In–band Registration*, tedy úplně stejná registrace, jakou provádíte při vytvoření účtu na Jabber serveru. Zajímavé ale je, že díky obecnému rozšíření XEP–0004 Data Forms, je možné vytvářet formuláře podobné HTML formulářům na webu. To může uživateli nabídnout vysoký komfort a přehlednost úprav, podobnou specializovanému programu.



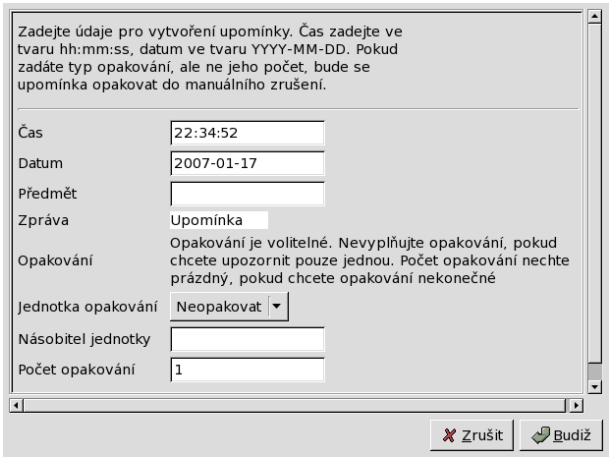
Obrázek 4.1: Ukázka zobrazení služeb v klientu Psi

4.2.3 Editace upomínky

K rozlišení jednotlivých úkonů jsem použil různých částí zdroje. Tím, že se jedná o jiné Jabber ID, lze při registraci rozlišit jaký formulář se má klientovi poslat. Pro zdroj `events` se posílá vždy formulář pro novou upomínku. Zdroj má popisek *Seznam upomínek*, při jeho prohlížení se vypíše seznam čekajících upomínek s jejich zkráceným popiskem a časem, kdy budou vyvolány. V Psi

verze 0.10 je velmi příjemné to, že na tuto položku stačí poklepat a otevře se nový formulář pro zadání upomínky. Ve formuláři je automaticky předvyplněné aktuální datum a čas klienta, s korekcí časového pásma ze strany robota. Po přijetí upomínky si robot zase převede čas na univerzální čas a teprve takový si uloží do databáze.

Ve formuláři je možné si také zvolit opakování upomínky. Je možné omezit počet opakování, po jeho vypršení se upomínka smaže. Pokud bude zvolen s popup nabídky typ opakování, ale počet opakování zůstane prázdný, bude upomínka brána jako nekonečná a periodicky se bude posílat až do zrušení nebo úpravy uživatelem. Příklad editačního formuláře je možné vidět na obrázku 4.2. Je na něm mimo jiné vidět malá kolonka pro editaci hlavního textu upomínky. Gajim chybně interpretuje víceřádkový vstup a kolonka vůbec neodpovídá tomu, jak by vypadat měla. Značný počet klientů má menší nebo větší problémy s rozvržením těchto formulářů. Dynamické formuláře bez pevně daných políček se používají krátkou dobu a většinou jde jeno o dvě nebo tři políčka při registraci k transportu.



Obrázek 4.2: Ukázka editace upomínky v Gajimu

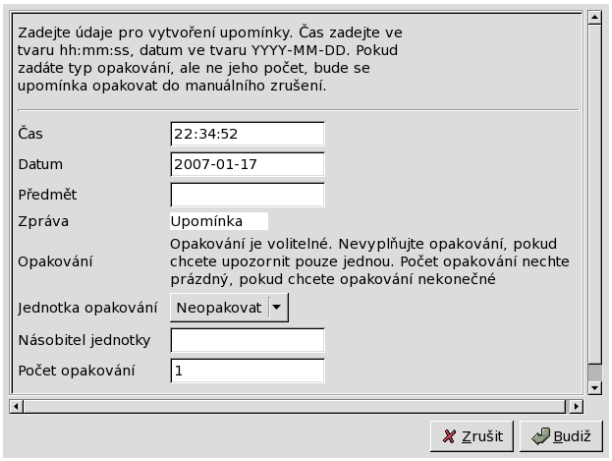
S mým formulářem nemělo velké problémy Psi ani Tkabber. Jak vypadá ten stejný formulář v Tkabberu je vidět na obrázku 4.3. Naopak Exodus sice formuláře umí, ale rozvržení je značně odpudivé. Jednotlivé kolonky mají okraje přes sebe, vedle nich je nevzhledné posouvání řádků. V Exodusu je asi pohodlnější použít editaci textovou, protože editace přes registrace je sice plně funkční, ale není vůbec uživatelsky příjemná.

Tkabber jako jediný z klientů obsahuje přímo v registračním formuláři tlačítko *Unregister*, které je možné použít k úplnému smazání upomínky. U ostatních klientů je nutné pro zrušení upomínky použít textový režim robota.

4.3 Textové příkazy

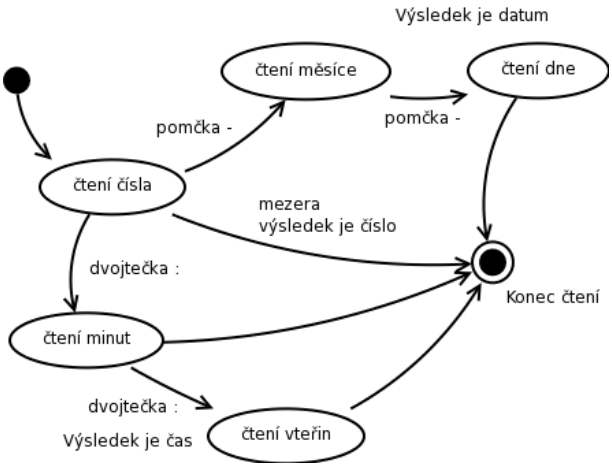
4.3.1 Lexikální analyzér

Robot obsahuje minimalistický lexikální analyzer pro rozlišování jednotlivých slov. Sice robot neumí žádné složité výrazy a většina zadávaných slov jsou slova klíčová, rozdělení na jenotlivé tokeny umožňuje snadno ignorovat mezery mezi slovy a poskytuje prostor pro snadnější další rozšiřování robota.



Obrázek 4.3: Ukázka editace upomínky v Tkabberu

Zatím má lexikální analyzér za úkol rozlišovat jenom interpunkční znaky, přečíst čísla a do uvozovek uzavřené řetězce. Dále rozlišuje čtení času a data ve formátu ISO 8601 i s kontrolou přípustných rozsahů. Ukázku stavového automatu vidíme na obrázku 4.4.



Obrázek 4.4: Ukázka stavového automatu při čtení čísla

4.3.2 Klíčová slova

Klíčová slova jsou interně převáděna ze slov na numerické konstanty. Numerické vyjádření přidá trochu rychlosti na zpracování výrazu, ale to není důvod jejich použití. Jako numerické hodnoty jsem je mohl využívat v konstrukci switch — case a snadněji tak nastavovat konečný automat.

Další velkou výhodou je možnost považovat několik tvarů jednoho slova za stejné klíčové slovo, takže v přecházení do různých stavů se musela napsat jenom jedna hodnota slova. Zvláště výhodný je takový přístup, pokud potřebuju mít několik tvarů toho stejného slova obslouženy jako slovo jediné. Například pro robota slovo *minuta, minut, minuty nebo minute* je pořád jeden výraz, a není důvod pro všechny tvary stejného významu zohledňovat syntaxi.

4.3.3 Syntaxe

Syntaktický analyzátor je stavový automat, který postupně upřesňuje parametry výsledné operace. Příkazů není příliš mnoho, pro práci s upomínkami je potřeba jenom několik málo operací. Použítím syntaktického analyzátoru jsem se pokusil trochu přiblížit lidské řeči. V lidské řeči používáme víc slov a méně speciálních znaků. Některá slova mají nepatrný význam a robot je může jednoduše ignorovat.

Cílem syntaktického analyzátoru je především umožnit rozvoj dalších parametrů a nových příkazů, které by se těžko přidávaly do speciálního čtení příkazů. I když jsem se snažil využít uživatelsky hezčích formulářů pomocí registrace, právě v možnostech dalších příkazů je větší prostor pro rozšiřování funkcí.

4.4 Omezení platformy

Protože zdaleka ne všechna volání knihoven jsou implementována na všech systémech stejně, potýkal jsem se při implementaci s přenositelností robota na více platforem.

Například chybí univerzální možnost překódování UTF-8 rětězce na znaky plně v unikódu. Pro Windows i POSIX neexistuje jednotné rozhraní.

Pro práci s časem jsem potřeboval intenzivně pracovat s mezinárodním časem. Knihovna jazyka C ale neumí převést datum vyjádřené v jednotlivých položkách na typ `time_t`. Jediné dostupné řešení je použití volání `mktime`, které ale pracuje pouze s lokálním časem, nikoliv univerzálním. Pro práci s univerzálním nabízejí BSD i GNU/Linux systémy volání `timegm`, které pracuje přímo s univerzálním časem.

I když většina Jabber/XMPP serverů může běžět na platformě Windows, všechny větší servery bývají obvykle na unixové platformě. Existuje velké množství podpůrných skriptů pro POSIXové systémy, ale je minimum pomocných programů pro správu serveru na platformách windows. Protože i můj server běží na systému GNU/Linux, zvolil jsem nasazení robota pouze na tomto systému. Tím můžu využít i některých volání, které na platformě windows vůbec nejsou.

Kapitola 5

Možnosti dalšího vývoje

5.1 Internacionalizace

Momentálně je robot napsaný v českém jazyce a s uživatelem komunikuje jenom česky. I když česká komunita kolem protokolu Jabber je poměrně aktivní, mnohem více uživatelů by využilo cizojazyčné verze jako jsou anglická, případně francozská. Původně jsem chtěl použít už v tomto robotu automatickou volbu jazyka a komunikaci s uživatelem v rodném jazyce. Překlad jsem chtěl zajišťovat pomocí rozhraní GNU gettext, ale při jeho zkoumání jsem zjistil problémy při použití na straně serveru. Tento systém sice umožňuje překlad programů do jiných jazyků a kompletní přizpůsobení jazykových vlastností, neumí ale přepínat rychle jazyky mezi sebou.

XMPP definuje použití `xml:lang` tagu pro automatickou detekci jazyka a novější servery, kterých neustále přibývá, už korektně přidávají atribut s jazykem ke každé odeslané zprávě nebo stavu. Automatická detekce jazyka a komunikace s uživatelem se přímo nabízí. Stejně jako je běžné, že webový server vám při návštěvě školních stránek ukaže českou stránku a angličanovi ukáže text v jazyce anglickém, budou snad brzy dostupné služby i v síti XMPP s podporou více jazyků. Je ale potřeba najít, nebo vytvořit, rozhraní pro překlad do jiného jazyka, které se může přepínat v jediném programu pro každého uživatele jinak.

5.2 Další možnosti

Úkonů, které může robot vykonávat, je velmi mnoho. Kromě zasílání upomínek by mohl spravovat třeba záložky na navštívené stránky, zajímavé odkazy s krátkým komentářem nebo obecné poznámky. Možnosti závisí především na srozumitelném zadávání požadavků. Například na serveru jabbim.cz už existuje robot na vyhledávání vlakového spojení, ale vyhledávání se musí uzavírat do nepříjemných hvězdiček a uvedení přestupní stanice je velice složité. S použitím jazyka blízkého lidskému by se podobné žádosti zadávaly intuitivně a jistě by přilákaly mnohem více uživatelů.

Další možnou volbou je rozšíření robota o sledovač stavu pro WWW stránky, protože robot stejně musí sledovat přítomnost uživatele. Ve spolupráci s rozšířením *User Geolocation* by mohl zaznamenávat i polohu uživatele a ukazovat na mapě. Problém je jenom s právy na mapové podklady, které získat pro další šíření není levná záležitost.

5.3 Webové rozhraní

I když se mi myslím podařilo dokázat, že s pomocí Jabber klienta je také možné pohodlně zadat upomínky do většinou slušně vypadajících formulářů, webové stránky mají stále své nesporné

výhody. Je to jejich obrovská rozšířenost a kompatibilita, kde se vám nestane moc často, že formulář nevypadá hezky. Internetový prohlížeč používá na internetu každý uživatel. Webový formulář lze lépe logicky organizovat a určit jeho vzhled i s rozvržením a velikostí jednotlivých položek.

Pokud by si chtěl někdo přesunout například všechny data narozenin do mého robota, rozhraní které poskytuje jabber klient neposkýtá takovou přehlednost, jakou lze snadno docílit na webové stránce pomocí dynamického jazyka. Protože robot využívá mysql databáze, propojení by bylo poměrně jednoduché a snadno funkční.

5.4 Synchronizace s kalendářem

Pro správu většího množství upomínek, nebo dokonce rozvrhu, se hodí možnost synchronizovat taková data se specializovanou aplikací, jako je například Microsoft Outlook. Při správě velkého množství upomínek se upravování jednotlivých položek stává dost těžkopádným, stejně jako zadání většího množství nových položek. Není ani možné zazálohovat si všechny upomínky pro případ výpadku služby.

Do budoucna by bylo užitečné umožnit import a export upomínek do stolního programu pro organizaci času. I když je protokol XMPP snadno rozšiřitelný, komfort užívání specializovaného programu nemůže nabídnou, ani nikdy nebude. I když s pomocí formuláře se dá zadání upomínky zpříjemnit, oproti přehlednému grafickému kalendáři je to stále velmi pracné. Pomocí synchronizace s programem kalendáře by bylo možné využít editaci více položek specializovaného prostředí a pro upozorňování na upomínky s výhodou využít sítě XMPP. Zvláště v kombinaci s externími branami do SMS sítě by upozorňování bylo velice užitečné.

Pro synchronizaci se stolní aplikací je možné použít například standardizovaný XML dokument, který by se pomocí přenosu souborů odeslal službě ke zpracování. Podobně by bylo možné zazálohovat si všechny již vytvořené upomínky a přenést je do stolní aplikace, například k vytištění rozvrhu. Jako vhodný formát se nabízí například formát iCalendar, který je podporován mnoha aplikacemi.

Kapitola 6

Závěr

Síť XMPP v sobě skrývá obrovský potenciál. Existuje celá řada služeb uživatelům této sítě. Od použití jednoduchého robota pro vyhledávání zadaných rozšíření podle jména po zajímavé webové mapy, ukazující stav a polohu uživatele. Můj příspěvek mezi další služby je robot schopný pracovat s poznámkami ve více časových pásmech.

Myslím si, že se mi povedlo zajímavým způsobem zužitkovat možnosti obecného formuláře pro zadávání upomínek. Každý ze specializovaných Jabber/XMPP klientů je bez problémů schopný zaregistrovat novou upomínku z hlavního prohlížeče služeb. Sice ne každý klient podporuje i úpravu existující upomínky pomocí registrace, ale z těch používanějších i tohle nedělalo problémy.

Velké překvapení pro mě bylo neexistence seznamu časových pásem a velké rozdíly v časech mezi jednotlivými státy. V různých částech světa se pro stejné zkratky používají úplně jiná časová pásma. Navíc zavedení letního času je přítomno pouze v některých státech a neexistuje jednotný algoritmus pro všechny státy. Určit tak spolehlivě kolik hodin je na určeném místě ve světě není vůbec triviální. Standardní knihovna jazyka C neposkytuje takové volání. Protože jména časových pásem nejsou standardizována žádnou mezinárodní institucí, jednoznačně určit posun času od univerzálního ani dost dobře nejde.

Bohužel při zkoumání možností současných klientů jsem zjistil, že podpora právě pro zjišťování času klienta není příliš široká. V některých případech sice funguje zjištění univerzálního času, ale pro fungování tohoto robota je zásadní určení běžného času na straně uživatele. Možná proto mnoho podobných robotů na internetu není, protože je potřeba zlepšit podporu na straně klientů. Jediný Tkabber je schopen sdělit svůj čas i se záznamem časového pásma tak, aby bylo možné automaticky časové pásmo zjistit. Často je tak nutné se spolehnout na ruční určení časového pásma, a tedy absenci automatického posunutí na letní čas a zpět.

Zajímavostí je klient Gajim. Jako jediný už podporuje novější rozšíření XEP-202 s číselným vyjádřením posunutí hodin. Malý problém spočívá v orientaci časového pásma. V jazyce python, ve kterém je klient napsán, vrací volání knihovny posunutí, jaké je třeba přičíst k lokálnímu času, abychom získali čas univerzální. Tento údaj je nesprávně vrácen jako odchylka časové zóny. Klient tak sice jediný protokol implementuje, ale je nutné s tímto údajem zacházet v rozporu s dokumentací k dosažení očekávaného výsledku. Na tuto chybu jsem autory upozornil a pravděpodobně bude brzy odstraněna.

Literatura

[1] DJ Adams. Xep-0009: Jabber-rpc. [online]. Last changes: 2006-02-09 [cit. 2007-01-18]. Dostupné z WWW `http://www.xmpp.org/extensions/xep-0009.html`.

[2] D. Reed J. Oirakiren. Internet realay chat protocol. [online]. Last modified: 24 May 1993. [cit. 2007-01-18]. Dostupné z WWW `ftp://ftp.rfc-editor.org/in-notes/rfc1459.txt`.

[3] Peter Saint-Andre. Xep-0016: Privacy lists. [online]. Last changes: 2006-11-27 [cit. 2007-01-18]. Dostupné z WWW `http://www.xmpp.org/extensions/xep-0016.html`.

[4] Peter Saint-Andre. Xmpp core. [online]. Last modified: 30 Sep 2004. [cit. 2007-01-18]. Dostupné z WWW `http://www.ietf.org/rfc/rfc3920.txt`.

[5] Peter Saint-Andre. Xmpp im. [online]. Last modified: 01 Oct 2004. [cit. 2007-01-18]. Dostupné z WWW `http://www.ietf.org/rfc/rfc3921.txt`.

[6] Webová stránka. Gajim. [online]. [cit. 2007-01-15]. Dostupné z WWW `http://www.gajim.org/`.

[7] Webová stránka. Psi. [online]. [cit. 2007-01-15]. Dostupné z WWW `http://www.psi-im.org/`.

[8] Webová stránka. Time zones. [online]. [cit. 2007-01-17]. Dostupné z WWW `http://www.timegenie.com/timezones.php`.

[9] Webová stránka. Xmpp extensions. [online]. [cit. 2007-01-15]. Dostupné z WWW `http://www.xmpp.org/extensions/`.

[10] Webová stránka. Zadevední letního času. [online]. [cit. 2007-01-14]. Dostupné z WWW `http://astrolot.cz/mimo/letnicas.html`.

[11] WWW stránky. Xml. [online]. Last modified 2006/09/11. [cit. 2007-01-20]. Dostupné z WWW `http://www.w3.org/XML/`.

[12] Dave Winer. Xml rpc. [online]. Poslední změny: 30. 6. 2003 [cit. 2007-01-18]. Dostupné z WWW `http://www.xmlrpc.com/spec`.