VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

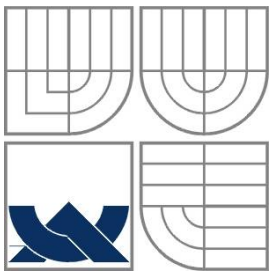
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JABBER KLIENT PRO PDA

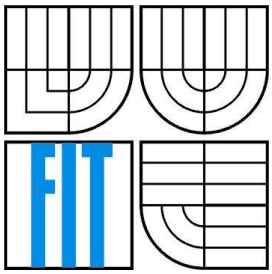
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JURAJ HRUDA
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JABBER KLIENT PRO PDA

JABBER CLIENT FOR PDA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JURAJ HRUDA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. MICHAL PAJGRT

BRNO 2008

Abstrakt

Náplňou tejto bakalárskej práce bolo vytvorenie knižnice pre komunikačný protokol Jabber a vytvorenie klienta ktorý prezentuje možnosti tejto knižnice. Knižnica aj klient by maly byť prevádzkované na platforme Windows Mobile ktorá je používaná na PDA, prípadne PDA kombinovaných s mobililným telefónom. Základom pre vytvorenie tejto práce bolo naštudovanie tejto platformy a taktiež aj verejne prístupného protokolu Jabber. Potom bolo potrebné si stanoviť čo všetko bude knižnica umožňovať a navrhnúť čo najlepšie rozhranie pre komunikáciu s knižnicou. Potom bolo možné pristúpiť k vlastnej implementácií. Knižnicu som sa rozhodol implementovať ako viacvláknovú aplikáciu. Knižnica je univerzálna a je jednoduché ju použiť pre implementáciu vlastného klienta.

Klíčová slova

Jabber, PDA, Windows Mobile, C#, komunikačný klient, viacvláknové aplikácie

Abstract

The main target of this Bachelor's thesis work was to create library for Jabber communication protocol and to create a sample client presenting library features. Both library and client should be used on Windows Mobile platform which is used on PDA and PDA combined with mobile phone. The basic thing in this work was to learn about Windows Mobile platform and also to learn the jabber protocol which is open. Then I had to find out what should the library have to do and design the best inteface for comunication with the library. Then the implementation should begin. I decided to implement library as multithread aplication. The library is universal and it's easy to create new clients with it.

Keywords

Jabber, PDA, Windows Mobile, C#, communication client, multithread applications

Citace

Hruda Juraj: Jabber klient pre PDA. Brno, 2008, bakalárska práca, FIT VUT v Brne.

Jabber klient pre PDA

Prohlášení

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

……
Juraj Hruda
13.5.2008

Poďakovanie

Zvláštne poďakovanie patrí Ing.Michalovi Pajgrtovi za odborné konzultácie, nápady pre vylepšie a korekcie v technickej dokumentácií.
Ďalej moje poďakovanie patrí zamestnávateľovi K2 Atmitec Brno za pochopenie časovej náročnosti potrebnej k vypracovaniu tejto práce.

Obsah

Úvod

Internetové komunikačné klienty vznikly z potreby ľudí komunikovať medzi sebou čo najrýchlešie na akúkoľvek vzdialenosť. Pred vznikom takýchto klientov sa používal predovšetkým e-mail, prípadne diskusné fóra. Komunikácia za pomoci týchto prostriedkov ale nebola bezprostredná a preto bolo potrebné vyvinúť nejaké nástroje za pomoci ktorých by bola možná okamžitá komunikácia. To viedlo k vytvoreniu komunikačných protokolov a následne klientov.

V dnešnej době je veľké množstvo týchto klientov a každý si môže vybrať to čo mu najviac vyhovuje. Najpožívanejšie sú v dnešnej dobe ICQ, AIM, MSN, Yahoo, Skype, Google talk a Jabber. Posledné dve spomenuté maju veľa spoločného pretože google talk je založený na protokole Jabber. Väčšina týchto klientov je komerčných, to znamená že sa v nich väčšinou objavuje reklama alebo ich nieje možné používať pre komerčné účely.

Jabber je otvorený komunikačný protokol založený na protokole XMPP. Poskytuje všetky služby ako ostatné komerčné protokoly. Základnou službou je výmena správ medzi 2 užívateľmi. Samozrejmosťou je dnes aj komunikácia medzi viacerými užívateľmi naraz v jednom okne, takzvaný skupinový chat. Medzi ďaľšie často používané služby medzi užívateľmi patrí napríklad prenos súborov, zobrazovanie avatarov(užívateľom zvolená podobizeň) a v neposlednej rade komunikácia pomocou hlasu(telefonovanie cez internet).

Implementácia v tejto práci prebiehala pre platformu Windows Mobile ktorá je požívaná na PDA. Najnovšia verzia systému windows mobile je v priebehu písania tejto práce 6.1. V tomto sytéme beží prostredie .NET 2.0 Compact Edition takže vývoj aplikácií je veľmi podobný ako pre stolné PC. Pre implementáciu bakalárskej práce bol použitý moderný programovací jazyk C# od firmy Microsoft.

V prvej kapitole sa budem venovať popisu platformy Windows Mobile a behovému prostrediu .NET 2.0 Compact Edition.

Druhá kapitola si kladie za cieľ objasniť problematiku pri implementácií viacvláknovej aplikácie, jej výhody a nevýhody.

Tretia kapitola popisuje podrobne protokol Jabber a jednotlivé služby ktoré tento protokol poskytuje.

V štvrtej kapitole je podrobnejšie popísaná vytvorená knižnica pre komunikáciu protokolom Jabber.

V poslednj kapitole je popísána implementácia ukážkového klienta používajúceho nami vytvorenú knižnicu.

1 Popis platformy Windows Mobile

Operačný systém Windows Mobile je určený pre mobilné telefóny a pre PDA. Najnovšia verzia v dobe písania tejto správy je 6.1. Existujú 3 verzie systému Windows Mobile a to Standard, Professional a Classic. Verzia Standard je určená pre mobilné telefóny, verzia Professional je určená pre PDA kombinované s mobilným telefónom a verzia Classic je určená pre PDA. Informácie o veziách systému sú prevzaté z [1].

Vývoj aplikácií pre Windows Mobile je najvhodnejší vo vývojovom prostredí Microsoft Visual Studio s nainštalovaným SDK pre Windows Mobile 6. Aplikácie sa následne spúšťajú v emulátore zariadenia zo systémom Windows Mobile alebo priamo v zariadení ktoré je pripojené k PC. Komunikácia zo zariadením prebieha za pomoci aplikácie ActiveSync ktorá slúži k synchronizácií s PC a k prenosu súborov.

Pri vývoji aplikácií je potrené mať vždy na pamäti to že každé zariadenie s týmto systémom môže mať iný displej. Zobrazenie môže byt orientované horizontálne alebo vertikálne. Aplikácia by teda mala byť schopná sa prispôsobyť rôznym natočeniam displeja a taktiež aj rôznym rozlíšeniam displeja. Väčšina zariadení používajúcich Windows Mobile majú dotykový displej a s tým je potrebné tiež počítať při vývoji aplikácií. Existujú však aj zariadenia bez dotykového displeja a s tým by mal programátor pri vývoji aplikácií tiež počítať.

1.1 Behové prostredie .NET 2.0 Compact edition

Behové prostedie .NET 2.0 Compact edition je určené pre zariadenia PDA a mobilné telefóny. Umožnuje vývoj aplikácií ktorý je veľmi podobný vývoju aplikácií pre stolný počítač. Je to umožnené tým že toto prostredie vzniklo zjednodušením prostredia .NET 2.0.

Prostredie .NET 2.0 Compact edition je nezávíslé na použitom hardware a je vďaka tomu možné vyvinúť jednu aplikáciu a bez akejkoľvek úpravy ju používať na rôznych zariadeniach. K vývoju aplikácií pod touto platformou je možné použiť rôzne jazyky a to Visual Basic, C#, a C++. Visual Basic a C++ sú staršie programovacie jazyky ktoré boli upravené a doplnené o nové vlastnosti. Jazyk C# bol špeciálne navrhnutý pre programovanie aplikácií pre .NET Framework.

Jadrom prostredia .NET je modul CLR čo je Common Language Runtime. Aplikácia sa prekladá vždy do medzikódu a CLR sa postará pri spustení aplikácie o preklad z medzikódu do strojového kódu danej platformy. To zaručuje nezávistosť aplikácie na zariadení na ktorom bude prevádzkovaná. Programátor tak nemusí vo väčšine prípadov dopodrobna poznať všetky hardwarové platformy na ktorých bude aplikácia prevádzkovaná a stačí mu poznať iba prostredie .NET ktoré sa postará o harwadovú kompatibilitu.

1.1.1 Jazyk C#

Jazyk C# je moderný programovací jazyk špeciálne vyvinutý pre prostredie .NET. Jazyk je plne objektovo orientovaný, to znamá že všetko s čím pracujeme je objekt. Syntax jazyka je založená na jazyku C rozšírená o objetový prístup. Informácie o jazyku C# som čerpal z [2].

Základným prvkom jazyka C# sú menné priestory. Slúžia na logické zoskupovanie rôznych tried do skupín. Pomáha to programátorovi sa lepšie orientovať v tom ktoré triedy spolu súvisia a na čo presne slúžia. Každý programátor si môže vytvárať svoje vlastné menné priestory a tým organizovať svoje triedy. Na začiatku každého zdrojového súboru C# programátor určuje ktoré menné priestory chce používať podobne ako sa napríklad v jazyku C vkladali hlavičkové súbory. Tým že si programátor vyberie menné priestory s ktorými bude pracovať získava prístup k triedam ktoré su definované v týchto menných priestoroch.

Jazyk je plne objektový a to znamená že všetko je považované za objekt. Aj základné dátové typy ako sú numerické dátové typy sú považované za objekt a je možné s nimi pracovať ako s objektami. Všetky objekty v C# sú zdedené z objetu System.Object. Vďaka tejto vlastnosti je možné napísať funkciu ktorá bude mať ako parameter akýkoľvek objekt. Každý objekt obsahu štadardne aj niektoré funkcie ako je napríklad Equals(), GetHashCode(), GetType() a ToString(). Každý objekt si môže tieto funkcie predefinovať aby sa chovali požadovaným spôsobom.

Vzhľadom na objetovú orientovanoť jazyka C# je tu vo veľkej miere používaná dedičnosť tried. To znamená že prevezmeme vlastnosti nejakej triedy a upravíme jej chovanie podľa vlastných potrieb. Môžme do triedy pridať svoje vlastnosti a funkcie alebo môžme prekryť už existujúce a upraviť tak ich chovanie. Je možné taktiež predefinovať funkciu operátorov pre porovnávanie a matematické operácie.

Ďaľšou vlastnosťou jazyka C# sú kolekcie ktoré slúžia pre uchovávanie viac jako jednej položky. Má podobné vlastnosti ako pole z dynamickou veľkosťou. Dalo by sa to prirovnať k lineárnemu zoznamu pretože kolekcia má svoj prvý prvok a je možné od dané prvku vždy pristúpiť k následujúcemu prvku. Základné kolekcie v C# sú ArrayList čo je reprezentácia dynamického poľa, zásobník, fronta, slovníkové a hešové tabuľky.

Rozšírením kolekcí sú generické kolekcie. Do štandardných kolekcií sa vkladá vždy objekt typu System.Object a je následne potrebné pretypovať ho späť na typ ktorý sme do kolekcie vložili. Nieje tým ale zaručená typová bezpečnost a môže tým dochádzať k chybám. U generických kolekcií sa už pri definícií určuje aký typ objektov budeme do kolekcie vkladať a tým sa zaručí typová bezpečnost a nieje nutné následné pretypovávanie objektov.

1.1.2 Výnimky

Za behu aplikácie môže dochádzať k chybám ktoré nemusia byť spôsobené programátorom. Môžu byť spôsobené chybným vstupom od užívateľa, chybou v prenose po sieti, chybou v niektorom zo súborov ktoré su editovateľné užívateľom atď. Pri tvorbe aplikácie je preto potrebné myslieť na to že k týmto chybám môže dochádzať a v prípade že je to možné tak je potrebné sa z touto chybou vhodným spôsobom vysporiadať.

Vždy v prípade keď dôjde k chybe tak je vyhodená výnimka. Tým je prerušený tok aplikácie v mieste kde sa nachádza a vykonávanie aplikácie sa ukončí v prípade že nieje výnimka zachytená aplikáciou ale až operačným systémom. Výnimky sa zachytávajú pomocou konštrukcie try...catch...finally. Catch môže mať definovaný jeden parameter a to výnimku ktorú daný blok zachytáva. Môže existovať niekoľko rôznych blokov catch a každý z nich môže zachytávať rôzne druhy výnimiek a reagovať na ne odlišným spôsobom. Bloky pre odchytávanie výnimiek môžu byť vnorené do seba a je možné výnimky prepúšťať do nadradeného bloku v prípade že si daný blok nevie s výnimkou poradiť. Blok finally sa vykoná vždy nakoniec aj v prípade že bola vyhodená výnimka. Je vhodné sem preto dávať napríklad rôzne deštruktovy alebo uvoľnit zablokované zdroje.

Každý programátor môže vytvárať vlastné výnimky zdedením z triedy System.Exception alebo System.ApplicationException. Programátor tak môže dať najavo z modulu ktorý vyvíja inému modulu ktorým je používaný že nastala chyba a je potrebné sa s ňou nejakým zpôsobom vysporiadať.

Pokiaľ nieje výnimka odchytená tak to zpôsobí to že sa dostane až k operačnému systému ktorý nemá inú možnosť ako zobraziť chybové hlásenie a okamžite ukončiť danú aplikáciu. To môže byť problém lebo v aplikácií môžu byť nejaké neuložené údaje.

1.1.3 Udalosti

Programovanie za pomoci udalostí funguje tým spôsobom že si zaregistrujeme funkciu ktorá sa má zavolať pri danej udalosti. Keď nastane daná udalosť tak sa zavolá obsluha udalosti ktorú sme si zaregistrovali. Môže to byť napríklad reakcia na užívateľský vstup z klávesnice alebo myši. Na jednu udalosť môže byť zaregistrované neobmedzené množstvo funkcií.

Predtým ako bol zavedený do programovacích jazykov princíp udalostí tak sa bolo potreba cyklicky pýtať operačného systému či nenastala nejaká udalosť, potom bolo treba zistiť aká udalosť bola vykonaná a podľa toho vykonať obsluhu. Zo zavedením udalostí je tento postup oveľa jednoduchší lebo iba nastavíme aká funkcia sa má zavolať pri danej udalosti.

Udalosti slúžia aj k synchronizácií rôznych vlákien a to tým spôsobom že nieje potrebné aby jedno vlákno čakalo na pokyn druhého vlákna ale bude sa vykonávať užitočný kód ktorý nieje nijak závyslí na operácií na ktorú by sa čakalo. Vo chvíli keď to jedno vlákno požaduje tak vyvolá udalosť ktorá v druhom vlákne spôsobí vykonanie danej operácie. Zefektívni sa tým kód tým že je možné odstránit zbytočné čakanie.

Programátor môže vytvárať svoje vlastné udalosti a registrovať si podľa potreby k nim obslužné funkcie. Užitočné je to hlavne v tom prípade ak aplikácia spravuje viac procesov alebo vlákien a je potrebné medzi nimi komunikovať a predávať si správy. Vlastnej udalosti môžme definovať ľubovoľné parametre funkcie ktoré si budeme pomocou danej udalosti predávať.

V prostredí .NET 2.0 Compact edition sa vlastné udalosti realizujú tým že sa vytvorí delegát, k nemu prislúchajúci event handler a funkcia ktorá bude vyvolávať danú udalosť. Delegát slúži ako ukazateľ na inú funkciu. Pri vytvorení delegátu sa definuje akú funkciu bude zastupovať a potom vždy pri vyvolaní daného delegátu sa vykoná funkcia ktorú zastupuje. Funguje teda ako callback funkcie v jazyku C kde sa k tomu používali ukazatele na funkcie. Event handler je v podstate zoznam delegátov ktoré sa zaregistrujú pre spracovanie danej udalosti. Funkcia vyvolávajúca danú udalosť zistí či je nejaký delegát zaregistrovaný pre obsluhu danej udalosti, v prípade že je tak sa vyvolajú postupne všetci delegáti ktorý sú zaregistrovaný k danej udalosti presne v tom poradí v akom boli k danej udalosti zaregistrovaný.

1.1.4 XML

XML je univerzálny značkovací jazyk pre všeobecné použitie.Je to rozšíriteľný jazyk, každý si môže definovať svoju vlastnú štruktúru XML súboru. Každý prvok v XML môže mať začiatočnú značku a koncovú značku. Značky je možné do seba ľubovoľným spôsobom vnorovať. Začiatočná značka môže v sebe obsahovať ľubovoľný počet atribútov, medzi začitočnou a koncovou značkou môže byť umietnený text reprezentujúci dáta. Každý XML súbor obsahuje práve jeden koreňový prvok, XML je textový formát súboru takže si ho môže ktokoľvek prečítať a tiež ho môže užívateľ ručne vytvárať a upravovať.

Prostredie .NET je úzko zviazané s formátom XML. Nielen že umožnuje vytvárať a spracovávať tieto súbory ale aj rôzne konfiguračné súbory .NET projektov sú ukladané vo formáte XML. Taktiež aj komentáre zdrojového kódu sú vo formáte XML.

K práci s XML súbormi slúži objekt XmlDocument z menného priestoru System.Xml. Tento objekt implementuje model DOM(Document object model) čo znamená že XML súbor sa prevedie na stromový objektový model. Prvky ktoré su vnorené do daného prvku sú uchovávané v kolekcií ChildNodes. Každý prvok obsahuje taktiež túto kolekciu, zároveň všetky prvky obsahujú kolekciu s atribútmi daného prvku.

Taktiež je možné na platforme .NET používať jazyk XPath pre vyhladávanie potrebných údajov v XML súbore. Slúžia k tomu objekty XpathDocument, XpathNavigator a XpathNodeIterator.

2 Viacvláknové aplikácie

Každú aplikáciu zavedie opračný systém pri jej spustení ako jeden proces a jedno vláko. Pre väčšinu aplikácií je to postačujúce, aplikácie ktoré všach chcú asynchrónne reagovať na rôzne požiadavky alebo chcú využívať výkonu viacerých procesorov musia fungovať ako viacvláknové aplikácie.

Pri tvorbe týchto aplikácií ale vzniká problém synchronizácie medzi vláknami lebo bežia súčastne pokiaľ nieje použitý niektorý zo synchronizačných mechanizmov pre pozastavenie niekorého z vlákien.

2.1 Význam viacvláknových aplikácií

Viacvláknové aplikácie majú v dnešnej dobe veľmi veľký význam hlavne z dôvodu využívania výkonu viacerých jadier prípade procesorov. V dnešnej dobe má už každý moderný procesor viac ako jedno jadro a aplikácie musia držat krok s vývojom hardware.

Bez toho aby bola aplikácia viacvláknová je v podstate nemožné využiť viac ako jedno jadro procesora z toho dôvoda že tieto jadrá bežia súčastne a program v počítači je postupnosť príkazov ktoré sa sekvenčne vykonávajú, to znamená že sa vykonávajú v presne určenom poradí. Programátor sám musí určiť ktoré časti programu sa môžu vykonávať súčastne bez toho aby bol narušený hladký beh programu. Tie časti aplikácie ktoré môžu bežať súčastne musí vhodne porozdeľovať do rôznych vlákien. Tento proces sa nazíva paralelizácia.

Programátor ale musí mať na pamäti že v systéme bežia aj iné aplikácie a nieje teda presne schopný určiť ktorá časť kódu z ktorého vlákna sa bude kedy vykonávať. Môže tým dochádzať k rôznym kolíziam na zdrojoch alebo používania nesprávnych hodnôt ak bude programátor predpokladať že už mu druhé vlákno predalo údaje ktoré požadoval ale pritom druhé vlákno ešte nestihlo jeho požiadavok spracovať. K riešeniu týchto problémov slúži synchronizácia vlákien.

2.2 Synchronizácia vlákien

Synchronizácia vlákien slúži k tomu aby rôzne vlákna nepristupovali k rovnakému miestu v pamäti v tom istom momente a taktiež aby v určitých prípadoch aby jedno vlákno počkalo na ukončenie operácie v inom vlákne. K synchnonizácií vlákien služia 3 spôsoby a to pomocou udalostí, mutexu a pomocou semafórov [3].

2.2.1 Udalosti

Synchnronizácia pomocou udalostí funguje tak že sa vlákna navzájom volajú pomocou udalostí. Funguje to tak že jedno vlákno si dá vypočítať alebo vykonať učitú operíciu druhým vláknom.

Medzitým vlákno ktoré zavolalo danú udalosť čaká na dokončenie dalej operácie. Týmto sa zamedzí tomu aby si rôzne vlákna navzájom upravovali pamät alebo aby vlákno čítalo dáta ktoré ešte niesu kompletné.

2.2.2 Mutex

Synchronizácia pomocou mutexu slúži k tomu aby nepristupovali rôzne vlákna naraz k jednomu zdroju. Väčšinou sa jedná o miesto v pamäti, môže se však jednať aj o nejaké hardwarové zariadenie, o riešenie kolízí na hardwarovom zariadení sa väčšinou stará operačný systém alebo ovládače. Funguje to tak že jedno vlákno požiada o mutex, v prípade že je daný mutex voľný tak vlákno ďalej pokračuje. V prípade že je mutex rezervovaný(využíva ho iné vlákno) tak sa vlákno pozastaví do chvíle kým nieje mutex uvoľnený druhým vláknom.

2.2.3 Semafóry

Synchronizácia pomocou semafórov funguje tak že umožnuje určenému počtu vlákien súčasný prístup k danému zdroju alebo časti kódu. Semafór sa dá použiť k dvom rôznym účelom.

Jeden z nich je ten že umožní blokovanie zdroja, zistenie či je zdroj voľný alebo je možné čakať na uvoľnenie daného zdroja.

Druhý dôvod pre využitie semafóra je využívanie obmedzeného počtu zdrojov. Funfuje to tak že každé vlákno požiada na počiatku o použitie daného semafóru. Interný čítač semafóru sa pri každej požiadavke zvýši pokiaľ nieje dosiahnuté maximum nastavené v danom semafóre. Ak je dosiahnuté maxim tak sa počká na uvoľnenie daného zdroja a potom pokračuje vykonávanie operácií v danom vlákne. Samozrejmosťou u semafórov je možnosť na otestovanie voľnosti zdroja, v prípade že je zdroj už plne obsadený je možné vykonávať inú operáciu a skúsiť použiť zdroj neskôr keď už by mohol byť voľný.

2.3 Okenné viacvláknové aplikácie

U okenných viacvláknových aplikácií napísaných v jazyku C# pre platformu .NET 2.0 Compact Edition môže nastať problém že je potrebné z jedného vláka vytvoriť okno a určitým spôsobom ho nastaviť a následne potom toto okno ovládať pomocou druhého vlákna. Problém je v tom že vlákno ktoré vytvorilo toto okno si ho zamklo spolu aj zo všetkými komponentami na formulári pre seba z dôvodu bezpečnosti aby nenastala situácia pri ktorej by viac vlákien naraz chcelo pracovať s jednou komponentou.

Riešením tohoto problému je vytvorenie delegátu ktorý bude spracovávať náš požiadavok pre prácu s oknom alebo komponentou na formulári. Ak sme vo vlákne ktoré vlastní danú komponentu tak môžeme priamo zavolať daného delegádata. V prípade že sa nachádzame v inom vlákne tak je

potrebné zavolať funkciu Invoke ktorá má ako parameter nášho delegáta. Táto funkcia sa postará o správne vykonanie kódu bez toho aby došlo ku kolízií pri prístupe ku komponente. Či je potrebné zavolať funciu Invoke(nachádzame sa vo vlákne ktoré nemá prístup k danej komponente) zistíme pomocou vlastnosti InvokeRequired. Týmto je daný problém jednoducho vyriešený.

Príklad kódu spracovania požiadavku pre okno z iného vlákna:

```
delegate void ReceiveMessageD();
private void ReceiveMessageCallback()
{
  …
}

if (Messages.InvokeRequired)
  Messages.Invoke(new ReceiveMessageD(ReceiveMessageCallback));
else
  ReceiveMessageCallback();
```

3 Protokol Jabber

Jabber je bezpečný otvorený decentralizovaný protokol pre výmenu správ medzi užívateľmi tejto siete [4]. Bezpečný je z toho dôvodu že je možné všetku komunikáciu šifrovať protokolom SSL. Protokol je verejne prístupný na internete a každý si teda môže spraviť svoj vlastný jabber server alebo klienta bez toho aby porušoval akékoľvek autorské práva. Jabber serverov je na internete veľké množstvo takže výpadok jedného serveru nespôsobí nikdy pád celej siete.

Každý užívateľ v tejto sieti je identifikovaný unikátnym JID(Jabber ID) ktorý je vo formáte užívateľ@server. Identifikátor je veľmi podobný emailovej adrese a veľmi ľahko by mohlo dôjsť k zámene, preto je vždy potrebné udať že sa jedná o JID.

Každý užívateľ môže byť prihlásený na jeden účet naraz z viacerých klientov. Identifikátor je potom zložený z troch častí a to užívateľ@server/zdroj. Každý zdroj by mal mať určenú číselnú prioritu. V prípade že je správa odoslaná na adresu užívateľ@server tak sa správa doručí implicitne klientovi ktorý je na zdroji s najväčšou prioritou. Pokiaľ chceme odoslať správu užívateľovi na daného klienta tak je možné odoslať správu na adresu užívateľ@server/zdroj.

Protokol jabber je založený na protokole XMPP(eXtensible Messaging and Presence Protokol) čo v preklade znamená „rozšíriteľný protokol pre prenos správ a informácií o dostupnosti užívateľov“.

3.1 XMPP

Protokol XMPP a protokol Jabber sú identické. Jednotlivé služby protokolu Jabber vznikli rozširovaním služieb poskytovaných protokolom XMPP. Všetky služby poskytované protokolom XMPP sú popísané na webových stránkach projektu. Protokol bol zverejnený organizáciou IETF.

Všetka komunikácia protokolu XMPP prebieha vo formáte XML. Komunikácia prebieha na porte 5222 alebo v prípade šifrovanej komunikácie na porte 5223. Celá komunikácia medzi dvoma uzlami siete XMPP môže byť chápaná ako dva XML súbory ktoré sú postupne vytvárané podľa potreby serverom a klientom. Koreňovým uzlom je vždy stream:stream čím vznikne nový XMPP stream ktorý je identifikovaný jednoznačným identifikátorom ktorý určí vždy server.

Komunikácia protokolom XMPP prebieha medzi serverom a klientom ale aj medzi dvoma rôznymi servermi. Servery medzi sebou komunikujú hlavne v tom prípade že chcú medzi sebou komuniovať dvaja užívatelia zaregistrovaný na rôznych serveroch.

3.1.1 Prihlásenie bez využitia SASL

Proces prihlásenia je popísaný v dokumente XEP-0078 [7]. Prihlásenie bez využiťia SASL je zastaralým spôsobom prihlásenia do siete jabber a nieje odporúčané. Implementácia tohto spôsobu

prihlásenia je však oveľa jednoduchšia a je serveromi z dôvodu spätnej kompatibility naďalej podporovaná. Z toho dôvodu tu popíšem aj tento spôsob prihlásenia.

Proces prihlásenia funguje na princípe IQ požiadavkov ktoré sú bližšie popísané v kapitole 3.1.4. Prihlásenie funguje tým spôsobom že server požiadame o prihlásenie, server nám odošle informáciu o tom aké polia je potrebné vyplnit, klient ich vyplní, odošle na server a ten overí prihlásenie a odošle klientovi informáciu o úspechu, prípadne neúspechu prihlásenia.

K prihláseniu slúžia dva rozlišné princípy a to „plaintext“ a „digest“. Server informuje klienta pomocou polí ktoré mu zašle o princípoch ktoré podporuje. Je na klientovi aby si zvolil vhodnejší princíp. Pri princípe „plaintext“ sa heslo prenáša v nešifrovanej podobe ako text, u princípu „digest“ je heslo prenášané len ako hash.

Je odporúčané použiť pri prihlásení bez SASL princíp „digest“ aby nedošlo k tomu že sa niekto iný dozvie heslo užívateľa. Je tu použítá hash SHA1 popísaná v dokumente RFC 3174. Hash je vytvorená tým spôsobom že spojí XMPP identifikátor daného spojenia a heslo ktoré používa užívateľ. Následne sa na tento spojený text aplikuje hashovací algoritmus a odošle sa serveru vo formáte hexa dát prostredníctvom malých písmen abecedy. Server spraví to isté, porovná hash ktorú dostal od klienta s tou ktorú si sám vytvoril a informuje klienta o úspešnom, prípadne neúspešnom prihlásení.

Príklad požiadavku klienta o autentifikáciu:

```
<iq type='get' to='shakespeare.lit' id='auth1'>
  <query xmlns='jabber:iq:auth'/>
</iq>
```

Príklad odpovede serveru na požiadavok o autentifikáciu:

```
<iq type='result' id='auth1'>
  <query xmlns='jabber:iq:auth'>
    <username/>
    <password/>
    <digest/>
    <resource/>
  </query>
</iq>
```

Príklad zaslania údajov k prihlásenie klientom serveru:

```
<iq type='set' id='auth2'>
  <query xmlns='jabber:iq:auth'>
    <username>bill</username>
    <digest>48fc78be9ec8f86d8ce1c39c320c97c21d62334d</digest>
    <resource>globe</resource>
  </query>
</iq>
```

Príklad odpovede serveru o úspešnom prihlásení:

```
<iq type='result' id='auth2'/>
```

Príklad odpovede serveru o neúspešnom prihlásení:

```
<iq type='error' id='auth2'>
  <error code='401' type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

3.1.2 Prihlásenie s využítím SASL

Prihlásenie pomocou SASL(Simple Authentificaion and Security Layer) je popísané v dokumente RFC 3920 [13]. Chýba v ňom však presný popis tvorby hashe ktorý sa používa pre autentifikáciu užívateľov. SASL je univerzálny nástroj používaný vo viacerých protokoloch pre zabezpečovanie komunikácie a autentifikáciu užívateľov.

V protokole XMPP je potrebné dať serveru najavo že chceme používať prihlasovanie prostredníctvom SASL namiesto pôvodného spôsobu prihlasovania používaného protokolom XMPP. Server o tom informujeme tým že do XML prvku ktorým sa vytvára stream pridáme informunáciu o verzií. Pokiaľ sa použije verzia minimálne 1.0 tak sa bude používať prihlasovanie za pomoci SASL. Server v tom prípade do odpovede o vytvorení nového XMPP streamu pridá ďaľšie údaje a to napríklad podporované spôsoby kompresnie údajov a podporované technológie pre autentifikáciu pomocou SASL.

Celá následujúca komunikácia prebieha zakódovaná v podobe Base64. Komunikácia je tvorená ako „challenge“ a „response“. Znamená to teda že nám server odošle požiadavku ktorú musíme dešifrovať a vytvoriť podľa nej hash pomocou ktorej server overí užívateľské meno a heslo.

Príklad odoslania požiadavky klienta o komunikáciu pomocou SASL:

```
<stream:stream
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       to='example.com'
       version='1.0'>
```

Príklad odpovede serveru zo zoznamom použiteľných mechanizmov autentifikácie:

```
<stream:stream
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       id='c2s_234'
       from='example.com'
       version='1.0'>
  <stream:features>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
      <mechanism>DIGEST-MD5</mechanism>
      <mechanism>PLAIN</mechanism>
    </mechanisms>
  </stream:features>
```

Príklad požiadavku o autentifikáciu daným mechanizmom:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
         mechanism='DIGEST-MD5'/>
```

Príklad odoslania „challenge“ serverom:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1dGgiLGNoY
XJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==
</challenge>
```

Obsah „challenge“:

```
realm="somerealm",nonce="OA6MG9tEQGm2hh",qop="auth",charset=utf-8,algorit
hm=md5-sess
```

Príklad odpovede klienta:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0iT0E2TUc5dEVRR
20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAwMDAwMDAxLHFvcD1hdXRoLGRpZ2
VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20iLHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE
1MjMyMWYyMTQzYWY3LGNoYXJzZXQ9dXRmLTgK
</response>
```

Obsah odpovede:

```
username="somenode",realm="somerealm",nonce="OA6MG9tEQGm2hh",cnonce="OA6M
HXh6VqTrRk",nc=00000001,qop=auth,digest-uri="xmpp/example.com",response=d
388dad90d4bbd760a152321f2143af7,charset=utf-8
```

Príklad kladnej odpovede serveru:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=
</challenge>
```

Obsah odpovede:

```
rspauth=ea40f60335c427b5527b84dbabcdfffd
```

Príklad odpovede klienta označujúci úspešnu autentifikáciu:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Odpoveď serveru v prípade že je autentifikáciu úspešne ukončená:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Ako je vidieť z príkladov tak všetok obsah prvkov „challenge“ a „respose“ je kódovaný vo formáte Base64 ktorý v sebe obsahuje komunikáciu SASL. V prvom „challenge“ odošle server klientovi všetky údaje potrebné k výpočtu hash ktorú klient použije ako súčasť odpovede. Klient k výpočtu hash potrebuje ešte meno a heslo užívateľa spolu náhodne vygenerovaných textom označeným ako „cnonce“. Postup pre výpočet hash ktorou klient odpovedá:

```
HA1_D    = MD5_BIN(login_name:realm:password)
HA1      = MD5_HEX(HA1_D:nonce:cnonce)
HA2      = MD5_HEX('AUTHENTICATE':digest-uri)
RESPONCE = MD5_HEX(HA1:nonce:nc:cnonce:qop:HA2)
```

Funkcia MD5_BIN vracia binárne dáta vytvorenej hash, funkcia MD5_HEX vracia hash vo forme textu ktorý vznikol prevedením binárných dát šestnástkovej sústavy. Po úspešnej autentifikácií

pomocou SASL nasleduje vytvorenie nového XML streamu v ktorom bude prebiehať celá ďalšia komunikácia.

3.1.3 Registrácia užívateľov

Registrácia užívateľov je definovaná v dokumente XEP-0077 [9]. Registrácia užívateľa slúži k tomu keď si chce užívateľ vytvoriť nový účet na serveri jabber. Povinné údaje k registrácií na danom serveri odosiela server klientovi na základe jeho prvotného požiadavku o registráciu. Spolu s tým server zasiela klientovi aj doplňkové informácie k registrácií ktoré by mali byť zobrazené užívateľovi. Klient následne tieto údaje vyplní a odošle na server. Server overí tieto informácie a odošle klientovi odpoveď. Dôvodom pre zamietnutie registrácie môže byť že užívateľ je už zaregistrovaný, server nepodporuje tento spôsob registrácie a preferuje registráciu webovým formulárom alebo server nepodporuje registráciu na danej úrovni zabezpečenia spojenia.

Ďalšou funkciou pre registráciu užívateľov je samozrejmosť možnosti zmeny hesla užívateľa. Okrem zmeny hesla protokol umožnuje aj odregistrovanie užívateľa zo serveru a v tom prípade sa uvoľní dané JID a môže sa naň zaregistrovať iný užívateľ. Všetka komunikácia funguje na princípe IQ požiadavkov popísaných v následujúcej kapitole.

Príklad požiadavky užívateľa o zaslanie povinných údajov od serveru:

```
<iq type='get' id='reg1'>
  <query xmlns='jabber:iq:register'/>
</iq>
```

Príklad odpovede servera:

```
<iq type='result' id='reg1'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      Choose a username and password for use with this service.
      Please also provide your email address.
    </instructions>
    <username/>
    <password/>
    <email/>
  </query>
</iq>
```

Príklad požiadavky od klienta pre registráciu užívateľa:

```
<iq type='set' id='reg2'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>Calliope</password>
    <email>bard@shakespeare.lit</email>
  </query>
</iq>
```

Príklad požiadavky pre zmenu hesla:

```
<iq type='set' to='shakespeare.lit' id='change1'>
  <query xmlns='jabber:iq:register'>
```

```
    <username>bill</username>
    <password>newpass</password>
  </query>
</iq>
```

Príklad požiadavky pre zrušene registrácie daného užívateľa:

```
<iq type='set' from='bill@shakespeare.lit/globe' id='unreg1'>
  <query xmlns='jabber:iq:register'>
    <remove/>
  </query>
</iq>
```

3.1.4 Stiahnutie zoznamu kontaktov

Každý užívateľ má na serveri uložený zoznam priateľov. Po každom prihlásení je odporúčané vyžiadať si od servera zoznam kontakov ale nieje to vyžadované. Vyžiadanie kontaktov je užitočné hlavne z toho dôvodu že užívateľ sa môže prihlasoť z rôznych klientov a počítačov, v prípade že by sa nevyžiadal zoznam môže dôjsť k nekonzistencií. Pokiaľ klient požiada server o zoznam kontaktov tak v tom prípade server následne informuje klienta o každej zmene vo viditeľnosti ostatných kontaktov, v prípade že oň nepožiada tak tieto informácie server nesmie zasielať. Proces sťahovania kontakov zo serveru je popísaný v dokumente RFC 3921 [8].

Požiadavky sú realizované na základe XMPP príkazu IQ(info/query). Tento príkaz funguje na princípe otázka a odpoveď. Príkaz je realizovaný pomocou XML prvku IQ ktorý má niekoľko atribútov. Tieto atribúty sú hlavne od akého užívateľa požiadavok pochádza, či sa jedná o otázku alebo odpoveď a identifikátor za pomoci ktorého sa k sebe páruje otázka a odpoveď. Prvok IQ obsahuje v sebe vnorený prvok, v prípade že sa jedná o otázku alebo odpoveď na otázku tak je týmto prvkom query. Tento prvok má atribút ktorý je pomenovaný „xmlns“ a jeho hodnota určuje o aký typ správy sa jedná. Výsledkom IQ príkazu je vždy odpoveď k danému požiadaku alebo informácia o chybe.

V tomto prípade keď sťahujeme kontakty zo servera tak typ požiadavku bude „get“ a druh otázky bude „jabber:iq:roster“. V prípade že nenastane žiadna chyba tak nám server vráti zoznam kontaktov, každý kontakt je samostatný prvok „item“ v xml ktorý má v sebe ešte vnorené skupiny do ktorých je kontakt zaradený. Všetky dôležité informácie o kontakte sú uložené ako atribúty prvku „item“.

Príklad požiadavku odoslaného klientom:

```
<iq from='juliet@example.com/balcony' type='get' id='roster_1'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```

Príklad odpovede od serveru:

```
<iq to='juliet@example.com/balcony' type='result' id='roster_1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='romeo@example.net'
```

```
            name='Romeo'
            subscription='both'>
        <group>Friends</group>
      </item>
      <item jid='mercutio@example.org'
            name='Mercutio'
            subscription='from'>
        <group>Friends</group>
      </item>
      <item jid='benvolio@example.org'
            name='Benvolio'
            subscription='both'>
        <group>Friends</group>
      </item>
    </query>
  </iq>
```

V odpovedi je vidieť že nám dal server niformácie o JID, mene a spôsobe zaregistrovania k informáciam o kontakte. Možné sú 4 možnosti:

- „none“ - užívateľ nevídí informácie o kontakte a taktiež ani kontakt nevidí informácie o klientovi
- „to“ - klient má prístup k informáciam o kontakte ale kontakt nemá prístup k informáciam o klientovi
- „from“ - klient nemá prístup k informáciam o kontkate ale kontakt má prístup k informáciam o klientovi
- „both“ - aj klient aj kontakt majú prístup k informáciám o druhej strane

3.1.5 Úpravy zoznamu kontaktov

Úpravy v zozname kontaktov sú popísané v dokumente RFC 3921 [9] rovnako ako sťahovanie zoznamu kontaktov zo serveru.

Užívateľ môže kedykoľvek pridávať, upravovať alebo mazať kontakty zo svojho zoznamu kontaktov na serveri. Vzhľadom na to že užívateľ môže byť v jednu chvíľu prihlásený k serveru z rôznych klientov musí v momente zmeny v zozname kontaktov server rozoslať zmeny všetkým klientom ktorý po prihlásení požiadali o zoznam kontaktov. Všetko je realizované pomocou požiadavkou IQ takže je vždy požadovaná odpoveď druhej strany na danú požiadavku.

Pri pridávaní užívateľa do zoznamu kontaktov je samozrejmosťou že sa chceme zaregistrovať k odberu informácií o viditeľnosti daného kontaktu. Pošleme teda serveru požiadavku o zaregistrovanie sa k odberu týchto informácií. Server zareaguje tým že znova rozošle všetkým klientom na ktorých sme prihlásený upravený kontakt. Nastaví sa tu príznak požiadavky o autorizáciu. Zároveň s tým sa odošle druhému kontaktu požiadavok o zdielanie informácií. V prípade že kontakt nieje v danej chvíli prihlásený tak sa požiadavok uloží a doručí sa vo chvíli keď sa kontakt prihlási. Kontakt následne povolí alebo zakáže zdielanie svojich informácií a server nám predá o tom informáciu. Server taktiež

rozošle všetkým klientom na ktorých sme prihlásený informáciu o zmene v zozname kontaktov. Zmena je v tom že sa zmení pole „subscribed“ podľa toho či klient povolil alebo zakázal zdielanie svojich informácií.

Rozosielajú sa tu dva druhy správ. Jedna z nich IQ query typu „jabber:iq:roster“ a druhým typom správ je „presence“. Tento typ bude popísaný ďalej v kapitole o viditeľnosti kontaktov.

Príklad príkazu od klienta pre pridanie kontaktu do zoznamu:

```
<iq type='set' id='set1'>
  <query xmlns='jabber:iq:roster'>
    <item jid='contact@example.org'
          name='MyContact'>
      <group>MyBuddies</group>
    </item>
  </query>
</iq>
```

Príklad odpovede od serveru kde rozosiela všetkým klientom na ktorých sme pripojený zmeny v zozname kontaktov:

```
<iq type='set'>
  <query xmlns='jabber:iq:roster'>
     <item jid='contact@example.org'
           subscription='none'
           name='MyContact'>
       <group>MyBuddies</group>
     </item>
   </query>
 </iq>

 <iq type='result' id='set1'/>
```

Príklad príkazu pre zaregistrovanie sa k odberu informácií o kontakte:

```
<presence to='contact@example.org' type='subscribe'/>
```

Zostávajúca komunikácia je podobná uvedeným príkladom, v komunikácií sa menia iba atribúty u XML prvkov.

3.1.6 Viditeľnosť kontaktov

Viditeľnost kontaktov je popísaná v dokumente RFC 3921 [9] rovnako údaje z predošlých dvoch kapitol. Viditeľnost kontaktov slúži k prenosu informáií o tom či sú kontakty prihlásené, prípadne či majú nastavený nejaký stav lepšie popisujúci ich prítomnosť. Preddefinované sú následujúce stavy prítomnosti:

- Online – užívateľ je prihlásený
- Away – užívateľ je dočasne neprítomný
- DND – užívateľ si nepraje byť rušený
- XA – eXtended away – užívateľ je neprístupný po dlhší časový úsek
- Chat – užívateľ by si rád s niekým písal

- Offline – užívateľ je momentálne odhlásený

K základným preddefinovaným stavom je si môže užívateľ definovať svoj vlastný textový popis ktorý chce aby ostatný videli. Tento popis je možné definovať v rôznych jazykových mutáciach.

Komunikácia pre výmenu informácií o viditeľnosti užívateľov je realizovaná pomocou XML príkazov „presence“. Za pomoci týchto príkazov je realizovaná aj registrácia užívateľov k odberu informácií o stave ostatných užívatelov.

V prípade že užívateľ chce zmeniť svoj stav tak odošle serveru informáciu o zmene svojho stavu a server automaticky rozošle túto informáciu všetkým užívateľom ktorý sú zaregistrovaný k odberu informácií o stave. K tomuto odberu sa užívatelia zaregistrujú automaticky tým že požiadajú server o zoznam kontaktov a dostávajú informácie od užívateľov od ktorých majú k tomu povolenie. Tým že klient požiada server o zaslanie zoznamu kontaktov mu server automaticky po zaslaní zoznamu kontaktov odošle aj infomácie o viditeľnosti všetkých kontaktov. Vďaka tomu užívateľ nemusí o tieto informácie žiadnym spôsobom žiadať a server ho o všetkom informuje automaticky.

Ďaľšou informáciu ktorá sa prenáša príkazom „presence“ je priorita daného klienta. Priorita je užitočná v tom prípade že je užívateľ pripojený naraz cez viacej rôznych klientov prípadne počítačov. Správy su potom odosielané na klienta z najväčšou prioritou pokiaľ nieje určené inak.

Príklad príkazu pre nastavenie stavu užívateľa:

```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cz'>Ja dvo&#x0159;&#x00ED;m Juliet</status>
  <priority>1</priority>
</presence>
```

Príklad príkazu od serveru ktorý informuje klienta o zmene alebo počiatočnom stave užívateľa:

```
<presence
       from='juliet@example.com/balcony'
       to='romeo@example.net/orchard'
       xml:lang='en'>
  <show>away</show>
  <status>be right back</status>
  <priority>0</priority>
</presence>

<presence
       from='juliet@example.com/chamber'
       to='romeo@example.net/orchard'>
  <priority>1</priority>
</presence>
```

V ukážkovom prípade odpovede od servera je prezentovaný príklad toho že užívateľ je pripojený zároveň z rôznych klientov, v každom z týchto klientov má nastavenú inú viditeľnosť a prioritu. V tomto konkrétnom prípade je prioritnejší klient „chamber“ kde je kontakt online a má nastavenú prioritu 1.

3.1.7 Prenos správ

Prenos správ je najzákladnejšou službou ktorú poskytujú internetové komunikačné klienty. Ak chce odoslať klient správu inému užívateľovi siete jabber pošle požiadku svojmu serveru, určí prijemcu a obsah správy. Server sa postará o to aby bola správa správne doručená druhému klientovi. Pokiaľ je cieľový užívateľ zaregistrovaný na tom istom servery ako odosielateľ správy tak je to jednoduché, server iba predá správu správnemu užívateľovi. V prídade že je užívateľ zaregistrovaný na inom serveri tak je potrebné aby si servery predali navzájom danú správu a cieľový server sa potom musí postarať o doručenie správy správnemu klientovi.

V prípade že nieje cieľový klient momentálne pripojený tak musí byť správa uložená na serveri a ten sa musí postarať o doručenie správy ihneď po pripojení klienta.

Ako už bolo spomenuté na začiatku kapitoly 3 tak identifikácia užívateľa sa skladá z dvoch povinných častí a to meno užívateľa a server. Tretím nepovinným parametrom je zdroj ktorý identifikuje klienta z ktorého sme pripojený. V prípade že odosielame užívateľovi prvú správu tak môžeme ale nemusíme identifikovať aj zdroj ktorému posielame správu. Pokiaľ ale už prebieha komunikácia tak by sme mali odosielať správy užívateľovi na zdroj od ktorého sme prijali správu. Deje sa tak z toho dôvodu že ak je druhý užívateľ pripojený z viacerých klientov naraz tak by nebolo správne ak by nám napísal z jedného klienta a od nás by dostával správy na druhom klientovi.

Každá správa by mala obsahovať príjemcu a telo správy. Medzi ďaľšie voliteľné parametre správy patrí nadpis správy, druh správy a prípadne jednoznačný identifikátor vlákna pomocou ktorého bude následne možné jednoznačne dohľadať históriu určeného rozhovoru.

Príklad správy:

```
<message
       to='romeo@example.net'
       from='juliet@example.com/balcony'
       type='chat'
       xml:lang='en'>
  <subject>I implore you!</subject>
  <subject
         xml:lang='cz'>&#x00DA;p&#x011B;nliv&#x011B; prosim!</subject>
  <body>Wherefore art thou, Romeo?</body>
  <body xml:lang='cz'>Pro&#x010D;e&#x017D; jsi ty, Romeo?</body>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
</message>
```

3.1.8 Vyhľadávanie užívateľov

Vyhľadávanie užívateľov je definované v dokumente XEP-0055. Je užitočné v dvoch rozdielnych situáciách a to je pri registrácií nového užívateľa pokiaľ chceme zistiť či nieje dané JID už používané, alebo je to užitočné aj v tom prípade že hľadáme určitého užívateľa siete ale nevieme jeho JID ktoré potrebujeme k tomu aby sme si ho mohli pridať do zoznamu kontaktov.

Komunikácia medzi serverom a klientom funguje v prípade vyhľadavania veľmi podobne ako v prípade registrácií užívateľa. Znamená to teda že klient odošle serveru informáciu o tom že by chcel vyhľadávať užívateľa, server mu odošle názvy polí za pomoci ktorých je možné vyhľadávať a užívateľ tieto polia vyplní. Hneď potom sa tieto údaje odošlú na server a ten vráti zoznam užívateľov vyhovujúcich vyhľadávacím kritériám alebo informuje klienta o tom že nebol nájdený ani jeden užívateľ ktorý by vyhovoval daným kritériám.

Príklad požiadavku klienta o zaslanie možných polí pre vyhľadávanie:

```
<iq type='get'
    from='romeo@montague.net/home'
    to='characters.shakespeare.lit'
    id='search1'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'/>
</iq>
```

Príklad odpovede od serveru:

```
<iq type='result'
    from='characters.shakespeare.lit'
    to='romeo@montague.net/home'
    id='search1'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <instructions>
      Fill in one or more fields to search
      for any matching Jabber users.
    </instructions>
    <first/>
    <last/>
    <nick/>
    <email/>
  </query>
</iq>
```

Príklad požiadavku klienta na vyhľadanie užívateľov:

```
<iq type='set'
    from='romeo@montague.net/home'
    to='characters.shakespeare.lit'
    id='search2'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <last>Capulet</last>
  </query>
</iq>
```

Príklad odpovede serveru zo zoznamom užívateľov vyhovujúcich kritériám:

```
<iq type='result'
    from='characters.shakespeare.lit'
    to='romeo@montague.net/home'
    id='search2'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <item jid='juliet@capulet.com'>
      <first>Juliet</first>
```

```
      <last>Capulet</last>
      <nick>JuliC</nick>
      <email>juliet@shakespeare.lit</email>
    </item>
    <item jid='tybalt@shakespeare.lit'>
      <first>Tybalt</first>
      <last>Capulet</last>
      <nick>ty</nick>
      <email>tybalt@shakespeare.lit</email>
    </item>
  </query>
</iq>
```

Príklad odpovede serveru v prípade že nebol nájdený ani jeden užívateľ splňujúci dané kritériá:

```
<iq type='result'
    from='characters.shakespeare.lit'
    to='romeo@montague.net/home'
    id='search2'
    xml:lang='en'>
  <query xmlns='jabber:iq:search'/>
</iq>
```

4 Popis implementácie knižnice

Knižnica pre komunikáciu protokolom Jabber je implementovaná v jazyku C# formou DLL. Vzhľadom na to že jazyk C# je objektovo orientovaný tak návrh aj implementácia bola tvorená objektovou formou. V knižnici sa nachádza niekoľko tried, hlavnou z nich je trieda JabberProtocol ktorá v sebe obsahuje všetky potrebné funkcie a triedy ktoré su nutné pre vytvorenie komunikačného klienta siete Jabber. Všetko je umiestnené v mennom priestore JabberProtokol. Okrem hlavnej triedy JabberProtokol sú v tomto mennom priestore umietnené aj triedy výnimiek ktoré su používané v celej knižnici.

Trieda JabberProtocol obsahuje v sebe niekoľko verejných funkcií ktoré slúžia klientovi ku komunikácií zo serverom. Prvá funkcia ktorú musí klient zavolať je funkcia pre prihlásenie, ďalej je to stiahnutie zoznamu kontaktov zo serveru, nastavenie viditeľnosti užívateľa, odoslanie správy danému užívateľovi, pridanie daného užívateľa do zoznamu kontaktov, registrácia nového užívateľa a odhlásenie klienta od siete Jabber.

Okrem týchto funkcií obsahuje trieda JabberProtocol udalosti na ktoré sa môže klientská aplikácia zaregistrovať. Najdôležitejšiou udalosťou je informácia o príjme správy od niektorého užívateľa siete Jabber. Ďalšou veľmi dôležitou vlastnosťou je zmena viditeľnosti niektorého z užívateľov v zozname priateľov. Ostatné udalosti sú väčšinou len informatívne, napríklad prihlasovanie užívateľa sa zkladá z niekoľkých častí, ukončenie každej z týchto častí má za následok vyvovanie určitej na ktorú môže klientská aplikácia reagovať napríklad zobrazením informácie užívateľovi o tom v ktorej fázi prihlasovania sa nachádzame.

4.1 Sieťová komunikácia

Sieťová komunikácia je u protokolu Jabber realizovaná prostredníctvom TCP/IP na porte 5222.

Sieťové spojenie zo serverom je naviazané pri volaní funkcia pre prihlásenie do siete jabber. Server na ktorý sa prihlasujeme sa zistí z prihlasovacieho mena vďaka tomu že je súčasťou JID. Ku komunikácií sa využívajú triedy TcpClient a NetworkStream.

Sieťová komunikácia je v tejto knižnici realizovaná asynchrónne, to znamená že vždy pri príjme dát od serveru operačný systém zavolá funkciu knižnice pre spracovanie dát. Táto funkcia použije XML parser pre spracovanie dát od serveru a podľa toho čo obsahuje sa zavolá príslušná funkcia ktorá spracováva daný druh požiadavku.

4.2 Spracovanie XML požiadavkov serveru

Pre spracovávanie požiadavkou od serveru musíme spracovať XML súbor ktorý nám bol zaslaný. Tento XML súbor však nieje vo veľa prípadoch kompletný a musíme preto pred použitím XML parseru zistiť či je súbor kompletný, prípadne ho nejakým spôsobom upraviť. Je to realizované tým spôsobom že zistíme či XML súbor obsahuje text '<stream:stream ' čo znamá že dáta obsahujú počiatok XMPP streamu ktorý trvá počas celého spojenia zo serverom, dáta teda určite neobsahujú ukončenie daného XML prvku a preto si ho ručne doplníme. V prípade že to tak nieje tak doplníme nový fiktívny koreňový prvok a to z toho dôvodu že server nám môže naraz odoslať viacero požiadavkov pre spracovanie, XML súbor však nesmie obsahovať viac ako jeden koreňový prvok, pridaním nášho nového sa tento problém vyrieši.

Každý príkaz od serveru je v samostatnom XML prvku a rôzne druhy požiadavkou musíme spracovávať odlišným spôsobom. Najpoužívanejším druhom správy je IQ, ďaľšie druhy správ sú „message“ ktorý ktoré je používaný na prenost správ, „presence“ ktorý je určený k prenosu informácií o viditeľnosti užívateľov, „challenge“ ktorý je používany k SASL autentifikácií a „success“ prípadne „failure“ určené k prenosu informácie o úspešnej, prípadne neúspešnej autentifikácií pomocou SASL. Dané požiadavky sa spracujú a podľa potreby sa následne volajú udalosti prípadne sa uchovávajú rôzne informácie pre ďaľšie spracovanie.

4.3 Práca s kontaktami

Zoznam kontaktov je uložený v objekte „roster“ ktorý je umiestnený v triede JabberProtokol. Objekt roster je z triedy ContactList ktorá je zdedená z generickej kolekcie slovník. Ako klúč v tejto kolekcií je použitý textový reťazec obsahujúci JID daného užívateľa siete, objekt typu Contact je použitý ako hodnota v tejto kolekcií.

Trieda Contact obsahuje v sebe všetky údaje ktoré potrebujeme vedieť o danom kontakte. Ide o jeho JID, meno, viditeľnosť, zoznam skupín v ktorom sa daný kontakt nachádza a zoznam správ ktoré sme od daného kontaktu prijali ale ešte neboli spracované. Trieda obsahuje 3 funkcie. Jedna z funckií slúži na pridanie správy od klienta do zoznamu uloženého v tomto objekte. Ďaľšou funkciou je načítanie najstaršej správy ktorá je vo fronte a posledná je funkcia ktorá slúži k zisteniu či fronta obsahuje správy alebo je prázdna.

4.4 Prihlásenie užívateľa

Prihlásenie je v knižnici implementované s využítím SASL aj bez jeho využitia. Implicitne je zvolené prihlasovanie za pomoci SASL, je možné ho vypnúť pomocou vlastnosti triedy JabberProtocol ktorá sa volá UseSASL.

Pre overovanie hesla a vytváranie hash je preferovaný spôsob SHA1 prípadne MD5, záleží to na tom či je používaný SASL alebo nie. V prípade že server nepodporuje tieto mechanizmy pre overovanie sa použije prenos hesla v nešifrovanej podobe čo môže byť bezpečnostným rizikom v prípade že by niekto odpočúval komunikáciu.

Do funkcie pre prihlásenie je rovnako ako do funkcie pre sťahovanie zoznamu kontaktov zabudovaný časový limit po ktorom je operácia považovaná za neúspešnú a je vyvolaná výnimka vypršania časového limitu. Tento limit je možné z klientskej aplikácie zmeniť funkciou SetTimeoutLimit, implicitne je limit prednastavený na 30 sekúnd.

Implementácia prihlásenia pomocou SASL bola dosť problematická z toho dôvodu že sa mi nepodarilo nájsť žiadny zdroj ktorý by presne popisoval tvorbu hash ktorá slúži pre overovania hesla. Nakoniec som si tento postup naštudoval zo zdrojového kódu jabber serveru jabberd z funkcie ktorá overuje túto hash.

4.5 Vyhľadávanie užívateľov

Implementáciu vyhľadávania užívateľov som zpočiatku začal implementovať ale pri testovaní som objavil zaujimavú vlastnosť. Ani jeden z najpoužívanejších serverov vyhľadávanie užívateľov nepodporuje tak som sa rozhodol ho z knižnice nakoniec odstrániť a nahradiť.

5 Popis implementácie klienta

Klientská aplikácia je implementovaná vo forme formulárovej aplikácie riadenej udalosťami. K aplikácií je pripojená knižnica pre komunikáciu protokolom Jabber. Objekt typu JabberProtocol je umiestnený v hlavnej triede aplikácie nazvanej Program a tento je objekt je následne prístupný z celej aplikácie. Aplikácia je rozdelená do niekoľkých formulárov.

5.1 Prihlasovacie okno

Prihlasovacie okno je hlavné okno aplikácie a zobrazí sa ihneď po spustení aplikácie. Formulár obsahuje 2 polia pre zadanie prihlasovacieho mena a hesla. Ďalej sú na formulári umiestnené 2 zaškrtávacie polia a to pre nastavenie automatického prihlásenia pri ďaľšom spustení aplikácie a uloženie nastavení pre ďaľšie spustenie aplikácie. Nastavenia sa ukladajú a načítavajú z konfiguračného súboru login.xml. Formulár je zaregistrovaný k udalostiam kližnice ktoré služia k informovaní užívateľa o stave prihlasovania ktoré sú následne zobrazované na formulári. Po úspešnom prihlásení na server sa toto okno uzavrie a zobrazí sa okno zo zoznamom kontaktov.

5.2 Zoznam kontaktov

Ihneď po spustení zoznamu kontaktov sa zavolá funkcia knižnice Jabber pre stiahnute zoznamu kontaktov. Po úspešnom ukončení tejto funkcie sa na formulári vytvoria najskôr komponenty pre vytvorenie skupín a následne sa vytvoria komponenty s kontaktami ktoré sa umiestnia do správnych skupín.

Pre vytváranie týchto komponent na formulári som si vytvoril vlastné triedy ktoré sú zdedené z triedy Panel. Do komponenty pre skupiny je automaticky pridávaný textový popis s názvom skupiny. Do komponenty pre kontakty je umietnený textový popis s menom kontaktu, obrázok popisujúci jeho viditeľnost a časovač ktorý slúži k preblikávaniu obrázku v prípade že užívateľovi prišla správa.

Formulár je zaregistrovaný k odberu udalostí príjmu správy a zmeny viditeľnosti v knižnici pre komunikačný protokol Jabber. V prípade že sa zmení viditeľnost niektorého z kontaktov tak sa vyhľadá na formulári príslušná komponenta a zmení sa obrázok ktorý užívateľa o tomto informuje. V prípade príjmu správy sa najskôr zistí či nieje spustené okno s komunikáciou s daným kontaktom. Ak je toto okno spustené predá sa správa tomuto oknu, v opačnom prípade sa na formulári vyhľadá príslušný časovač ktorý ovláda preblikávanie obrázkov identifikujúcich prichádzajúcu správu a tento časovač sa spustí.

Z formulára zoznamu kontaktov je taktiež možné ovládať viditeľnost užívateľa, pridávať a odstraňovať užívateľov zo zoznamu kontaktov a premenovávať ich. Taktiež je z tohoto formulára prístupná funkcia pre zmenu hesla.

5.3 Okno správ

Okno správ slúži ku komunikácií s daným užívateľom. Na formulári sú umiestnené dve komponenty. Jedna z nich slúži k zobrazovaniu celej komunikácie a druhá slúži k písaniu nových správ. Formulár má funkciu na príjem správ ktorá je volaná z okna kontaktov. Táto funkcia robí iba jednu vec a to je to že zobrazí prichádzajúcu správu na formulári.

5.4 Okno pre pridanie nového užívateľa

Okno pre pridanie nového užívateľa do zoznamu kontaktov obsahuje 3 polia ktoré je potrebné vyplniť. Jedná sa o JID, meno užívateľa pod ktorým si prajeme ho mať uloženého v zozname a skupina do ktorej si ho prajeme pridať. Po vyplnení týchto údajom sa zavola funkcia knižnice JabberProtocol pre pridanie nového kontaktu do zoznamu. Táto funkcia po úspešnom pridaní užívateľa na server zavolá udalosť pomocou ktorej sa pridá nový kontakt aj do okna s kontaktami.

5.5 Okno pre registráciu nového užívateľa

Okno pre registráciu nového užívateľa je prístupné z prihlasovanie okna aplikácie. Formulár je rozdelený do 3 záložiek. Na 1. záložke sú umiestnené najdôležitejšie údaje o kontakte. Na 2. záložke sú umietnené menej dôležité údaje a na 3. záložke je iba textové pole ktorým sa môže užívateľ lepšie popísať. Samotná registrácia následne funguje tak že sa zavolá funkcia knižnice JabberProcol ktorý formulár informuje o úspešnom, prípadne neúspešnom prihlásení.

6 Záver

Výsledkom tejto práce je komunikačná knižnica protokolu Jabber. Spôsob implementácie klienta s využitím tejto knižnice je prezentovaný v ukážkovom Jabber klientovi.

Pri tvorbe tejto knižnice som získal veľmi užitočné zkúsenosti a vedomosti o tvorbe knižníc a sieťových aplikácií v prostredí .NET. Taktiež som sa zdokonalil v programovaní v jazyku C# a programovaní objetovo orientovaných aplikácií. Hlbšie som prenikol do problematiky tvorby viacvláknových aplikácií a riešeniu problémov ktoré vznikajú pri ich implementácií aplikáciách.

V priebehu tvorby knižnice pre protokol Jabber som prehodnotil svoj postoj k Open Source komunite a uvedomil som si jej silu. Vývoj knižnice bol vďaka tejto komunite veľmi jednoduchý pretože je na internete veľké množstvo užitočných zdrojov a ľudí ochotných a schopných poradiť.

Knižnica je plne postačujúca pre tvorbu ľubovolného klienta siete Jabber. V budúcnosti by bolo možné túto knižnicu rozšíriť o ďaľšie funkcie ako je prenos súborov, kompresia sieťovej prevádzky alebo skupinová komunikácia s viacerými užívateľmi naraz.

Ďaľším užitočným rozšírením funkčnosti tejto knižnice by bolo pridanie šifrovania sieťovej komunikácie pomocou SSL. V súčasnej verzií .Net Compact Edition však nieje táto funkčnosť k dispozícií narozdiel od prostredia .NET 2.0 určeného pre PC a bude pridaná pravdepodobne do ďaľších verzií tohto prostredia.

Literatúra

[1] WILSON, Jim. What's New for Developers in Windows Mobile 6 [online]. 2007 , 5/5/2008 [cit. 2008-05-08]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/bb278115.aspx>.

[2] NAGEL, Christian, et al. C# 2005 : Programujeme profesionálně. [s.l.] : [s.n.], c2006. 1398 s. ISBN 80-251-1181-4.

[3] SELVAM, R.. Thread Synchronization for Beginners [online]. c2004 [cit. 2008-05-08]. Dostupný z WWW: <http://www.codeproject.com/KB/threads/Synchronization.aspx>.

[4] Extensible Messaging and Presence Protocol [online]. c2007 [cit. 2008-05-08]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Jabber>.

[5] XML [online]. [cit. 2008-05-08]. Dostupný z WWW: <http://en.wikipedia.org/wiki/XML>.

[6] Jabber/XMPP Protocol Namespaces [online]. 2003 [cit. 2008-05-08]. Dostupný z WWW: <http://www.xmpp.org/registrar/namespaces.html>.

[7] SAINT-ANDRE, Peter. XEP-0078: Non-SASL Authentication [online]. 2003 [cit. 2008-05-08]. Dostupný z WWW: <http://www.xmpp.org/extensions/xep-0078.html>.

[8] SAINT-ANDRE, Peter. Extensible Messaging and Presence Protocol (XMPP) : Instant Messaging and Presence [online]. 2004 [cit. 2008-05-08]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc3921.txt>.

[9] SAINT-ANDRE, Peter. XEP-0077: In-Band Registration [online]. 2003 [cit. 2008-05-08]. Dostupný z WWW: <http://www.xmpp.org/extensions/xep-0077.html>.

[10] SAINT-ANDRE, Peter. XEP-0055: Jabber Search [online]. 2002 [cit. 2008-05-08]. Dostupný z WWW: <http://www.xmpp.org/extensions/xep-0055.html>.

[11] EASTLAKE, D., JONES, P.. US Secure Hash Algorithm 1 (SHA1) [online]. 2001 [cit. 2008-05-08]. Dostupný z WWW: <http://tools.ietf.org/html/rfc3174>.

[12] Simple Authentication and Security Layer [online]. [cit. 2008-05-08]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer>.

[13] SAINT-ANDRE, Jones. Extensible Messaging and Presence Protocol (XMPP): Core [online]. 2004 [cit. 2009-05-08]. Dostupný z WWW: <http://www.ietf.org/rfc/rfc3920.txt>.

Príloha – Obsah priloženého CD

Na priloženom CD nájdete kompletné zdrojové kódy knižnice a ukážkovej aplikácie. Pre ich spustenie je potrebné mať nainštalované Microsoft Visual Studio 2008 s nainštalovaným SDK pre Windows Mobile 6. Súčasťou CD je aj úplná technická dokumentácia včetne licencie a zadania.