Katedra Informatiky

Fakulta matematiky, fyziky a informatiky

Univerzita Komenského, Bratislava

Systém pre zdieľanie súborov v počítačových sieťach

(bakalárska práca)

Attila Mészáros

Študijný odbor: 9.2.1. informatika

Vedúci práce: RNDr. Jaroslav Janáček Bratislava 2008

Poďakovanie

Ďakujem vedúcemu bakalárskej práce RNDr. Jaroslavovi Janáčkovi za vedenie, odbornú pomoc a užitočné pripomienky. Ďalej by som rád poďakoval rodičom a všetkým ostatným, ktorí prispeli k dokončeniu tejto bakalárskej práce.

Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne, s použitím uvedenej literatúry.

Bratislava jún 2008

Abstrakt

Obsahom tejto bakalárskej práce je návrh a implementácia kompaktného systému, ktorý poskytuje informácie o zdieľaných súboroch používateľa, možnosť stiahnutia súborov povolenými osobami. Jeden z hlavných aspektov problematiky je bezpečnosť, bezpečná komunikácia medzi klientmi, s vylúčením akejkoľvek intervencie. Poskytovanie prístupových práv k zdieľaným súborom.

Kľúčové slová: bezpečná komunikácia, šifrovaný prenos, peer-to-peer, zdieľanie súborov.

Obsah

1. Úvod..1

2. Proces tvorby systému..2

2.1. Špecifikácia...2

2.2. Vývoj...5

2.2.1. Návrh..5

2.2.2. Realizovanie návrhu..5

2.2.2.1. Návrh klientskej aplikácie..6

Dátový model...7

Diagramy tried..9

Diagramy tried jadra systému a popis ich prevádzky...11

Reprezentovanie klientov..15

2.2.2.2. Návrh serverovej aplikácie...16

2.2.2.3. Komunikácia...17

Komunikačné protokoly..17

Pripojenia..17

Komunikačný most..19

2.2.3. Implementácia...23

2.2.3.1. Výber vývojového prostredia...24

2.2.3.2. Databázy..24

2.2.3.3. Bezpečnosť a komunikácia..25

2.2.3.4. Komunikačné XML protokoly....28

2.2.3.5. Hľadanie klientov na lokálnej sieti....29

2.2.3.6. Grafické používateľské rozhranie....30

2.2.4. Možnosti v budúcnosti....31

2.2.5. Záver....33

3. Použitá literatúra....34

4. Prílohy....34

1. Úvod

Žijeme v dobe, keď využitie a význam počítačových sietí je obrovský a neustále rastie. Počítačové siete sú využívané vo veľkých inštitúciách aj v bežných domácnostiach. Používanie internetu sa stalo súčasťou každodenného života človeka v práci aj vo voľnom čase.

Postupné rozšírenie a vývoj počítačových sietí a počítačových technológií, ako je zrýchlenie sietí, procesorov, rast kapacity pevných diskov atď. majú svoj vplyv na istý charakter týchto sietí. Jedným z dôsledkov vývoja je, že sa dostávajú do popredia takzvané „peer to peer“ technológie. Princíp a význam týchto technológií možno zhrnúť do jednej vety tak, že nie sú vybudované v zmysle klasickej klient-server architektúry, ale v architektúre typu klient-klient. V rámci tejto práci sa zaoberáme navrhovaním a implementáciou systému takéhoto charakteru s účelom zdieľania súborov. Budeme sa zaoberať problémami, ktoré sa týkajú bezpečnosti, problémami, ktoré sa vyskytujú pri komunikácii v takýchto sieťach; na vytváranie softvéru sa pozrieme aj z hľadiska softvérového inžinierstva.

2. Proces tvorby systému

Vývoj softvérového systému je proces, ktorý prechádza určitými etapami, majúcimi za cieľ zabezpečiť jeho funkcionalitu, kvalitu, spoľahlivosť. Tieto sú všeobecné požiadavky, a preto sú charakteristické pre všetky projekty. Medzi jeho základné etapy patria:

1. špecifikácia softvérového produktu,
2. vývoj (návrh a implementácia) softvérového projektu,
3. validácia softvérového projektu,
4. evolúcia softvérového projektu.

V nasledujúcom uvedieme popis prvých dvoch etáp a ich aplikáciu na náš softvér, ako aj techniky použité pri aplikácii.

2.1. Špecifikácia

Špecifikácia je aktivita, ktorá slúži na popis požiadaviek na produkt. Cez túto fázu prechádza každý softvérový produkt, či si to tvorcovia a zadávatelia uvedomujú alebo nie. Jej vedomé naplnenie však môže urýchliť vývoj softvérového produktu a zabrániť nezrovnalostiam tak pri vývoji, ako aj pri samotnom používaní.

Výstupom špecifikácie je dokument, ktorý by mal popisovať funkcie projektu, údaje, s ktorými produkt pracuje, ostatné obmedzenia na neho kladené, teda nie funkčné požiadavky. Špecifikácia požiadaviek je záväzná pre zadávateľa aj pre riešiteľa.

Popis požiadaviek je možné urobiť viacerými spôsobmi. Môže sa urobiť popis neformálne, použitím prirodzeného jazyka, semiformálne s použitím čiastočne formalizovaných jazykov (napr. UML) a formálne s použitím jazykov s matematickým základom. Čím je špecifikácia formálnejšia, tým je presnejšia, ale zároveň menej zrozumiteľnejšia pre laika. V našom prípade špecifikácia bola zadaná neformálne s prirodzeným jazykom. Náš systém má umožniť bezpečné zdieľanie súborov v počítačových sieťach. Má mať nasledovné funkcionality:

1. Systém má byť čo naj všeobecnejší, čo najmenej závislý na type počítačovej siete.
2. Systém má byť multiplatformový, teda čo najmenej závislý na type operačného systému.
3. Systém má umožniť presne stanoviť prístupové práva k zdieľaným súborom
    a. súbor má všeobecné práva pre neregistrovaných klientov,
    b. súbory sú radené do skupín, súbory majú svoje vlastné prístupové práva pre svoju skupinu, každý registrovaný klient môže byť zaradený do ktorejkoľvek skupiny.
4. Umožniť registráciu nových klientov, manažovať zaradenie registrovaných klientov do skupín.
5. Komunikácia, poskytovanie informácií o zdieľaných súboroch musí byť bezpečná, teda systém musí vylúčiť možnosť akejkoľvek intervencie zo strany tretej osoby.
6. Umožniť stiahnutie súborov klientom, ktorých prístupové práva to umožňujú. Stiahnutie súborov má byť tiež bezpečné.

7. Riešiť komunikačné problémy vyskytujúce sa pri neverejných IP adresách.
8. Systém má byť ľahko rozšíriteľný.
9. Systém má poskytovať ľahko zrozumiteľné a použiteľné grafické rozhranie používateľovi.

2.2. Vývoj

Vývoj môžeme rozdeliť na dve nezávislé fázy, na návrh a na implementáciu.

2.2.1. Návrh

Pri návrhu určíme, ako bude softvérový projekt realizovaný, pri návrhu tiež prechádzame iteratívne rôznymi etapami. Vo všeobecnosti:

1. architektonický návrh – rozdelenie na subsystémy,
2. návrh štruktúry subsystémov – rozdelenie na moduly,
3. návrh modulov – rozdelenie na triedy, respektíve na funkcie.

V rámci návrhu robíme tiež:

1. návrh používateľského rozhrania,
2. návrh štruktúry databázy,
3. prípadne stanovenie použitých algoritmov.

2.2.2. Realizovanie návrhu

Návrh systémov sa dá robiť všeobecne nezávisle od programovacieho jazyka. Pri našom návrhu sme použili všeobecný modelovací jazyk UML, ktorý je nezávislý od implementačného jazyka, i keď v našom prípade bol výber programovacieho jazyka už na začiatku, pri prezeraní špecifikácie, jednoznačný.

Pri špecifikácii jedným z kritérií bola multiplatformovosť a bezpečnosť. Z hľadiska multiplatformovosti prichádzajú do úvahy jazyky Java a C/C++. Ak by sme

zvažovali len toto hľadisko, bolo by možné uvažovať aj nad nejakým webovským programovacím jazykom, ako napr. PHP. Keďže však rozprávame o klientskej aplikácii, PHP nám jednoznačne nevyhovuje. Z viacerých dôvodov z dvoch programovacích jazykov sme zvolili Javu, nakoľko dá sa v ňom elegantne a jednoducho programovať úplne nezávisle od platformy, nepotrebujeme rekompiláciu kódu ako v prípade inej platformy, napr. ako pri C/C++. Okrem toho Java nám poskytuje veľmi silné sieťové prostriedky aj z hľadiska kryptológie, ako aj nástroje na vývoj grafického rozhrania.

Keďže ide o pomerne malý projekt, oproti vypracovaniu napr. operačného systému nie je nutné dodržiavať úplne presný postup. Po prečítaní špecifikácie bolo hneď zrejmé, že systém môžeme rozdeliť do dvoch subsystémov:

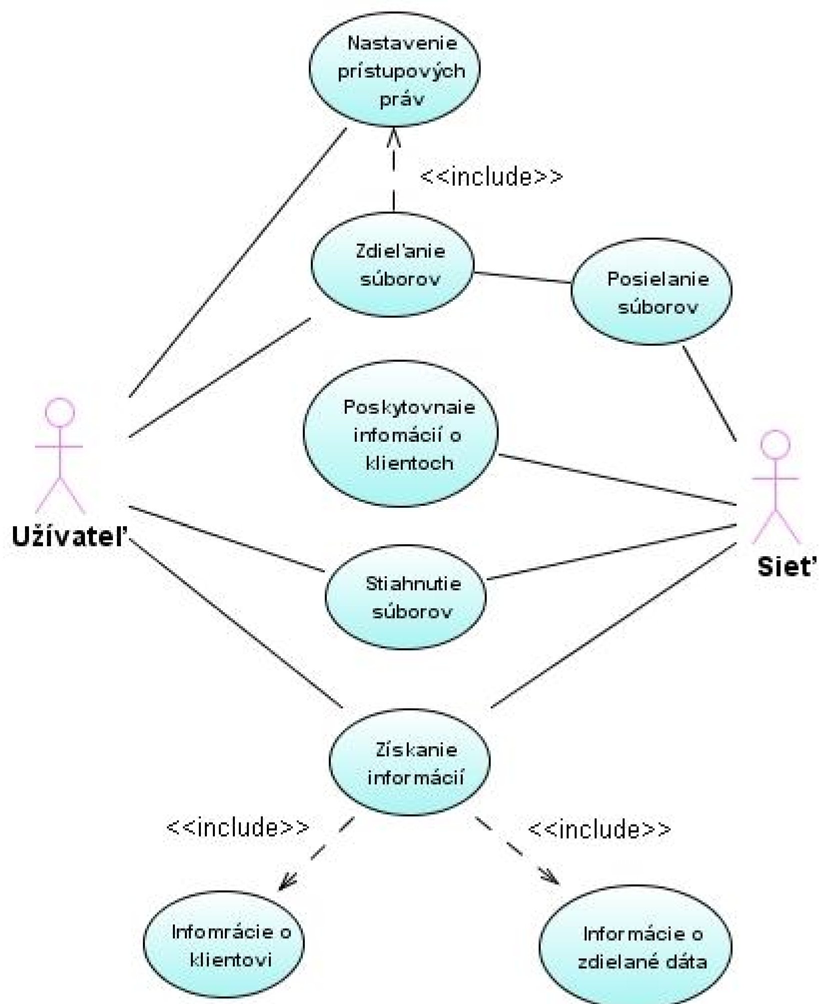
1. Klientská aplikácia – čo bude jadrom celého systému, má realizovať funkcionalitu zdieľania súborov, komunikáciu.
2. Serverová aplikácia – je pomerne jednoduchá pomocná aplikácia, na ktorú sa môžu klienti pripojiť a získavať informácie o iných klientoch.

V ďalšom si zvlášť rozoberieme návrh týchto dvoch subsystémov, a na konci niektoré komponenty, ktoré majú spoločné črty z hľadiska návrhu aj implementácie.

2.2.2.1. Návrh klientskej aplikácie

Na znázornenie funkcionality klientskej aplikácie slúžia diagramy prípadov použitia (Use Case Diagrams), ktoré sú výhodné aj z dôvodu, že sú zrozumiteľné aj pre bežných užívateľov.

Dátový model



Obr. 1: Diagram prípadov použitia pre klientskú aplikáciu.

Na diagrame je znázornené, ako užívateľ pristupuje k aplikácii a ako aplikácia komunikuje ďalšou aplikáciu cez počítačovú sieť. Pomocou tohto diagramu už máme jasnejší pohľad na to, čo všetko by malo poskytovať grafické rozhranie aplikácie, vidíme lepšie logiku nášho subsystému.

Dátový model

V ďalšom kroku bolo našou úlohou navrhnutie dátového modelu alebo databázy. Dátový model musí obsahovať nasledujúce informácie:

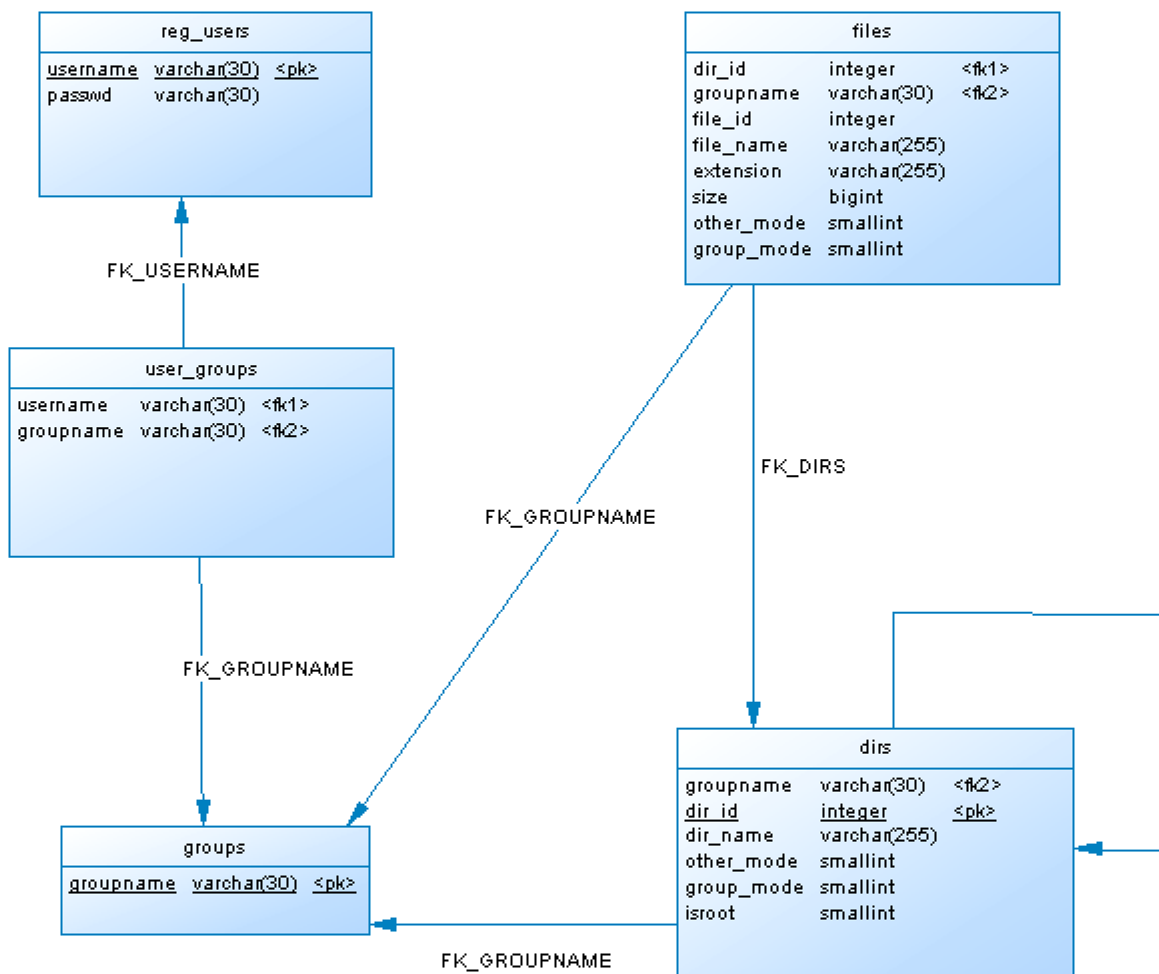
- údaje o skupinách (groups) – meno (name) skupiny,
- údaje o registrovaných užívateľoch (reg_users) – meno (name), heslo užívateľa (password),
- údaje o tom, ktorý užívateľ do akej skupiny patrí,
- údaje o zdieľaných súboroch.

Informácie o zdieľaných súboroch sme reprezentovali dvoma tabuľkami. Vytvárali sme stromovú štruktúru pomocou dvoch tabuliek: prvá obsahovala údaje o adresároch, druhá o súboroch. Každá z nich mala svoj primárny kľúč (file_id, dir_id) a cudzí kľúč (foreign key) na kľúč rodiča, (parent_id). Ak adresár nemá svojho rodiča, jeho rodič je reprezentovaný špeciálnou hodnotou, za ktorú sme volili 0 (nie NULL). Tabuľka adresárov aj súborov ešte obsahuje informácie o tom, aké má adresár prístupové práva a do akej skupiny patrí (groupname), čo je cudzí kľúč ukazujúci na tabuľku groups a v ňom na meno skupiny (name). Skupiny majú individuálne prístupové práva podľa súboru, resp. adresára; na realizovanie tejto funkcionality slúži ďalší stĺpec (group_mode). Používateľ, ktorý nepatrí do skupiny, má určené prístupové práva v ďalšom stĺpci (other_mode). Uvažovali sme aj nad tým, aby prístupové práva skupiny boli statické, takže každá skupina by mala stanovené prístupové práva, ktoré sú všeobecné pre všetky súbory, ktoré do tejto skupiny patria, ale súčasné riešenie je oveľa všeobecnejšie.

Pri adresároch je jeden stĺpec pre meno adresára, pri súboroch meno bolo reprezentované dvoma stĺpcami (file_name, extension). Pri súboroch bol pridaný ďalší stĺpec, v ktorom sme uchovali informácie o veľkosti súborov v bajtoch.

Obe tabuľky majú ďalší stĺpec na uchovanie informácie o tom, či je pri zdieľaný adresár, resp. súbor koreňový (root). Koreňovými adresármi, resp. súbormi nazývame tie, ktoré sú dostupné užívateľom priamo, teda sú na prvej úrovni v

reprezentačnej stromovej štruktúre zdieľaných dát.



Obr. 2: Dátový model.

Diagramy tried

Diagramy tried sú jadrom objektovo orientovaného modelovania. Znázorňujú typy objektov v systéme, ich vlastnosti a vzťahy medzi nimi. Pri návrhu väčších systémov ich takmer nie je možné vynechať. Pomocou nich sa už rovno dajú vygenerovať isté časti kódu. Pri navrhovaní diagramov musíme dávať pozor na to, aby sme náš model vytvorili tak, aby nebol závislý na implementácii – teda

definujeme triedy, verejné metódy tried a ich funkcionalitu bez toho, aby sme vedeli, ako budú pri implementácii realizované. Pri takomto spôsobe navrhovania počítame s nepredvídateľnými implementačnými problémami. Dobrým príkladom môže byť, ako to budeme vidieť nižšie, trieda AbstractNormalClient, ktorá reprezentuje klienta, s ktorým komunikujeme cez nejakú sieť, no v tejto fáze ešte presne nevieme, ako naše protokoly komunikácie medzi klientmi budú vyzerať, alebo ako bude komunikácia prebiehať. Vieme ale, že budú isté informácie, ktoré nám klient bude poskytovať, ako sú napr. zdieľané súbory. Preto dopredu vieme, že trieda bude obsahovať metódu isPassive(), čo nám vráti informáciu o tom, či klient má alebo nemá verejnú IP adresu.

Pri konkrétnom návrhu tried je dôležité, aby sme aplikáciu rozdelili do troch logických častí:

- používateľské rozhranie,
- funkcionalita,
- dáta.

Tieto tri logické časti by mali byť reprezentované príslušnými triedami. Triedy, implementujúce funkcionalitu aplikácie, pracujú nad dátovým modelom, teda nad inštanciami tried, ktoré reprezentujú dáta. Používateľské rozhranie zase pracuje s interfejsom, ktoré mu poskytujú triedy funkcionálnej časti. Dôležité je, aby funkcionalita nebola závislá na používateľskom rozhraní, teda k funkcionálnej časti môže patriť aj viac nezávislých používateľských rozhraní.

V ďalšom navrhujeme a analyzujeme hlavné architektonické záležitosti návrhu klientskej aplikácie pomocou diagramov tried.

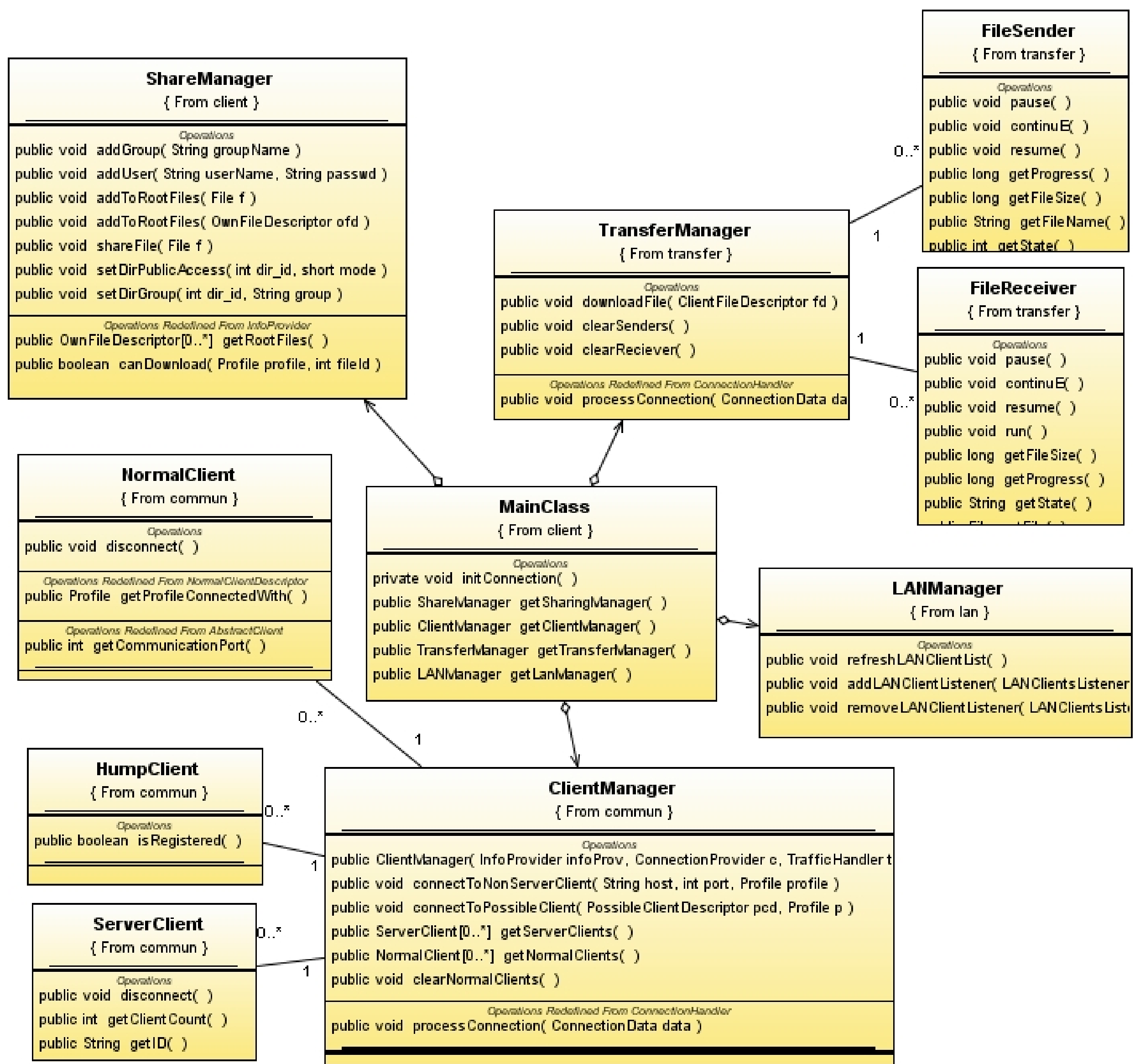
Diagramy tried jadra systému a popis ich prevádzky

Pri navrhovaní diagramov tried sme sa snažili o rozdelenie aplikácie do modulov. Pod modulom v našom prípade myslíme skupinu tried, ktorá reprezentuje istú časť funkcionality aplikácie. Moduly od seba nie sú nezávislé, práve naopak: z funkcionality systému vyplýva, že medzi sebou musia komunikovať. Väčšina v tomto zmysle chápanej komunikácie prebieha prostredníctvom takzvanej „listener" architektúry, čo je štandardná architektúra použitá v štandardných objektových knižniciach v Jave.

Hlavná idea tejto architektúry je, že každá trieda, ktorá potrebuje byť informovaná o nejakej udalosti, si implementuje interfejs, v ktorom sú definované metódy zavolané v prípade, že nastane príslušná udalosť. V triede, v ktorej udalosť nastane, je uchované pole týchto interfejsov. Keď nastane udalosť, je zavolaná príslušná metóda na všetky interfejsy v poli. Samozrejme, do pola treba pridať všetky triedy, ktoré budú informované. Toto nazveme registráciou. Výhodou tejto architektúry je nezávislosť, t.j. informované môže, ale rovno nemusí vedieť, kto ho oznámil, a trieda, ktorá informuje, nevie, koho informuje.

Samozrejme, keď z nejakej udalosti vyplýva striktne determinovaný dôsledok, volané sú rovno metódy modulu.

V nasledujúcom predstavíme hlavné, takzvané jadrové moduly našej aplikácie a ich prevádzku.



Obr. 3: Diagramy tried jadra (zoznam metód nie je kompletný).

Jadro systému tvorí trieda MainClass a triedy priamo k nej viazané. Slúži na úplné rozdelenie funkcionality od grafického rozhrania. Tu sú inicializované všetky ostatné moduly. Grafické rozhranie si od nej môže vypýtať všetky moduly, s ktorými môže ďalej pracovať. V opačnom smere teda informovanie o udalostiach grafického rozhrania je realizované prostredníctvom už spomenutej „listener“ architektúry. MainClass je singleton, teda taká trieda, ktorá je inštanciovaná len raz.

Pri inštancovaní modulov im odovzdáme potrebné prostriedky, ktoré od nás

vyžiadajú; sú to interfejsy, cez ktoré môžu pracovať s databázou, alebo trieda, ktorá implementuje aktuálny spôsob pripojenia.

Štyri hlavné moduly reprezentujú triedy ShareManager, ClientManager, TransferManager a LanManager.

ShareManager je najjednoduchší zo všetkých modulov, tvorí ho jedna trieda, v ktorej realizujeme väčšinu operácii nad databázou. Ideou je, že všetky operácie nad databázou robíme na jednom mieste, je to dôležité kvôli synchronizácii dotazov. Aby sme neponúkali zbytočné operácie jednotlivým modulom, ShareManager implementuje interfejsy, v ktorých sú definované operácie potrebné pre moduly.

LanManager je trieda, ktorá slúži na nájdenie klientov na lokálnej sieti. Od nej si môžeme vypýtať zoznam týchto klientov, prípadne si ju žiadať o aktualizáciu tohto zoznamu. Touto triedou sa budeme zaoberať ešte podrobnejšie.

ClientManager slúži na vykonanie všetkých operácií, pokiaľ ide o pripojenia na ďalších klientov. Jeho hlavnými komponentmi sú polia inštancií triedy NormalClient, ServerClient a HumpClient. Tieto triedy sú zodpovedné za sieťovú komunikáciu, poskytujú nám operácie nad jednotlivými klientmi, skrývajú implementáciu komunikácie aj všetky nízkoúrovňové protokoly. Všetky operácie sú transparentné, teda pracujeme s nimi, ako keby títo klienti boli na tom istom počítači. ServerClient reprezentuje pripojenie na server, NormalClient priame pripojenie na ďalšieho klienta, HumpClient prichádzajúce pripojenie, teda klienta, ktorý je pripojený na nás.

ClientManager si uchová dynamické pole zvlášť pre všetky typy klientov.

TransferManager má na starosti prenášanie súborov. Triedy FileSender a FileReciever realizujú posielanie a stiahnutie súborov. TransferManager nám poskytuje interfejs, cez ktorý sa dostaneme k funkcionalite stiahnutia súborov.

Na znázornenie prevádzky ClientManager-a a TransferManager-a uvádzame dva príklady. Prvým je znázornenie toho, ako prebieha pripojenie na ďalšieho klienta, a druhým to, ako prebieha posielanie súboru na žiadosť. Vybrali sme tieto dva príklady preto, lebo všetky ostatné funkcionality ClientManager–a, ako pripojenie na server alebo zaobchádzanie s prichádzajúcimi pripojeniami, resp. TransferManager–a, ako stiahnutie súborov, sú realizované takmer identickým spôsobom.

Funkcionalitu pripojenia na ďalšieho klienta implementuje ClientManager. Poskytuje verejnú metódu connectToNonServerClient(String host, int port, Profile p), ktorá je volaná pomocou používateľského rozhrania. S parametrami „host“ a „port“ definujeme adresu klienta, v parametri Profile definujeme, akým používateľským menom a heslom sa pripájame. Pomocou aktuálnej implementácie interfejsu Connector, ktorý je zodpovedný za vytvorenie všetkých pripojení v systéme, vytvoríme spojenie s klientom, pošleme správu s informáciami o našom klientovi, napr. či je pasívny, jeho meno, heslo. Posielanie prebieha pomocou statickej metódy triedy ConnectionMediator. Táto trieda ukrýva implementáciu posielania správ, a navvyše sa takto v niektorých prípadoch vyhneme duplikovaniu kódu. Hneď, ako sme túto správu poslali, vytvoríme triedu NormalClient s požadovanými parametrami, ktoré už v tomto momente máme k dispozícii. Tento objekt má na starosti všetky ďalšie komunikácie. Všetky potrebné informácie o zdieľaných súboroch, a právach mu poskytujeme cez interfejs InfoProvider, ktorý dostane ako parameter. Metódy InfoProvidera implementuje ShareManager. Komponenty používateľského grafického rozhrania, ktoré sú zaregistrované u ClientManager-a, sú informované o zmene stavu tak ClientManager-a, ako aj jednotlivého klienta. Používateľ vykonáva operácie ďalej už na príslušnom NormalClient objekte pomocou používateľského rozhrania (napr. zrušenie pripojenia).

Posielanie súborov má na starosti TransferManager. Keď žiadateľ vytvorí s nami pripojenie, jeho prvá správa (v našom prípade správa o žiadosti poslať súbor) je analyzovaná a posielaná ďalej TransferManager-ovi na ďalšie spracovanie. TransferManager si po spracovaní správy vytvorí inštanciu triedy FileSender, a potom, podobne ako v predchádzajúcom príklade, všetky ostatné komunikácie má na starosti tento objekt.

Reprezentovanie klientov

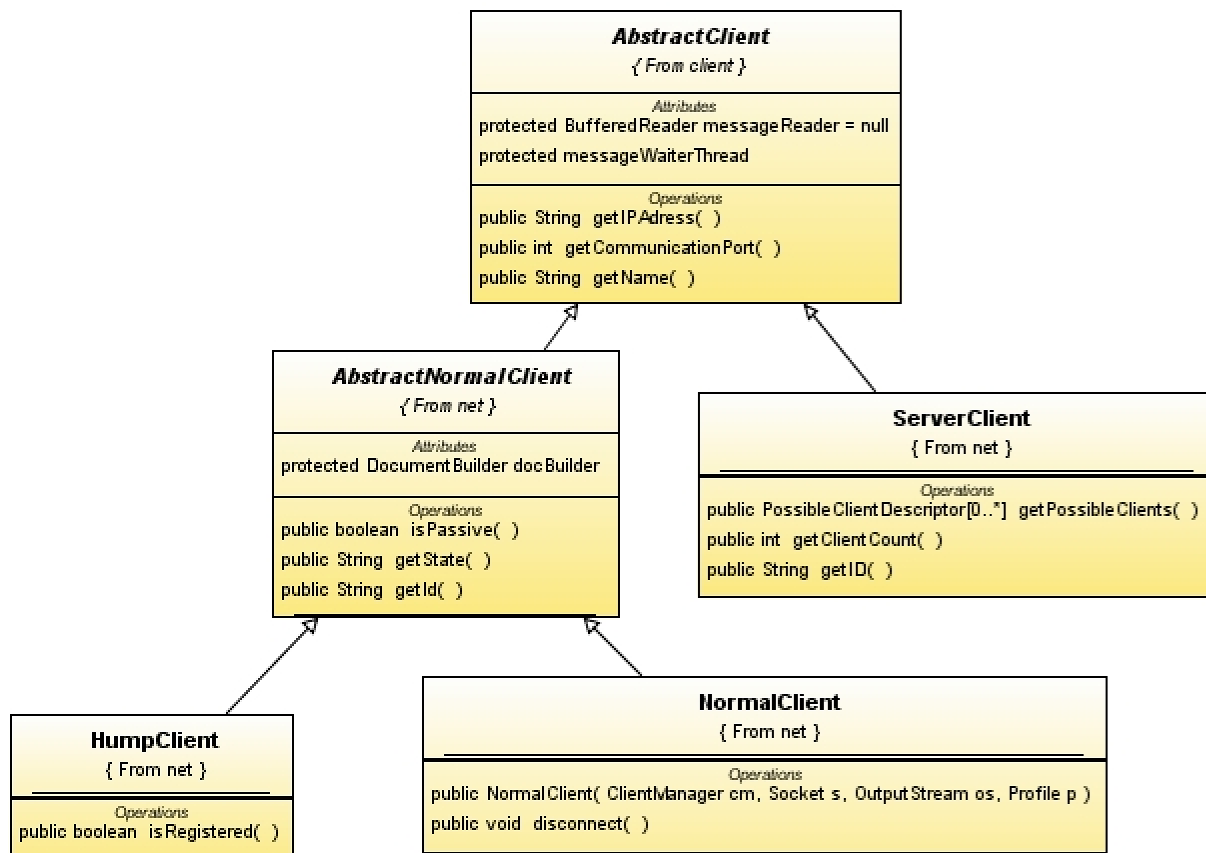
Jednotlivých klientov reprezentujeme triedami podľa toho, o aký typ klienta ide. Pod pojmom klient v tomto prípade máme na mysli buď klientské alebo serverové aplikácie.

Ideou pri návrhu týchto tried bolo dosiahnuť, aby sme s klientmi mohli pracovať tak, aby sieťové komunikácie, vlákna, synchronizácie vlákien boli skryté, teda pracujeme s nimi offline spôsobom – ako keby tá komunikácia ani neexistovala.

Triedna hierarchia pri reprezentácii klientov je veľmi pekným príkladom dedičnosti objektovo orientovaného programovania. Postupne si rozširujeme triedy podľa funkcionality. AbstractClient, ako to aj jej meno naznačuje, je abstraktná trieda, ktorá obsahuje informácie spoločné pre všetky typy klientov. Od nej dedí trieda ServerClient, ktorá reprezentuje server na ktorý sme, resp. boli pripojení.

Pri reprezentácii ďalšieho klienta sme najprv navrhli len jednu triedu, i keď musíme rozoznať, či sme sa na klienta pripojili my, alebo on na nás. To by ešte nebola veľká prekážka, s pridaním jednej boolovskej premennej si to ľahko vieme zapamätať. Lenže neskôr, už pri implementácii, sa ukázalo, že táto architektúra by nám veľmi skomplikovala život, jednak tým, že sa implementujú celkom odlišné protokoly, a potom tým, že sú inicializované iným spôsobom a v inom čase. Preto sme sa pri súčasnej architektúre rozhodli, že obidva typy klientov dedia od triedy AbstractNormalClient, ktorá obsahuje spoločné atribúty, ako ID klienta, alebo či je klient pasívny a sú už v nej implementované spoločné funkcionality, napr. posielanie

správ, synchronizácia posielania. Od nich dedí HumpClient, trieda reprezentujúca klienta, ktorý sa na nás pripojil, a NormalClient, na ktorého sme sa pripojili my.



Obr. 4: Hierarchia tried klientov (zoznam metód nie je kompletný).

2.2.2.2. Návrh serverovej aplikácie

Ďalšiu časť systému tvorí serverová aplikácia. Úlohou servera je poskytovanie informácie o klientoch. Na to, aby sme sa pripojili na klienta, musíme poznať jeho IP adresu a port, na ktorom čaká na pripojenie. V našom prípade tieto informácie poskytuje server. Samozrejme, úloha servera sa dá rozšíriť podľa toho, aké informácie o klientoch poskytuje, napr. o informácie o zdieľaných súboroch, a

tiež o ďalšie funkcionality, napr. o vyhľadávanie medzi súbormi priamo na serveri.

Ďalšou úlohou servera je riešenie problémov týkajúcich sa pripojenia. S touto problematikou sa budeme ešte zoberať v ďalších kapitolách.

Návrh serverovej aplikácie, čo sa týka architektúry tried, je veľmi podobný, takmer identický s návrhom architektúry klientskej aplikácie. Môžeme povedať, že návrh serverovej časti systému je zjednodušená forma návrhu klientskej aplikácie. Preto ho to ani neuvádzame.

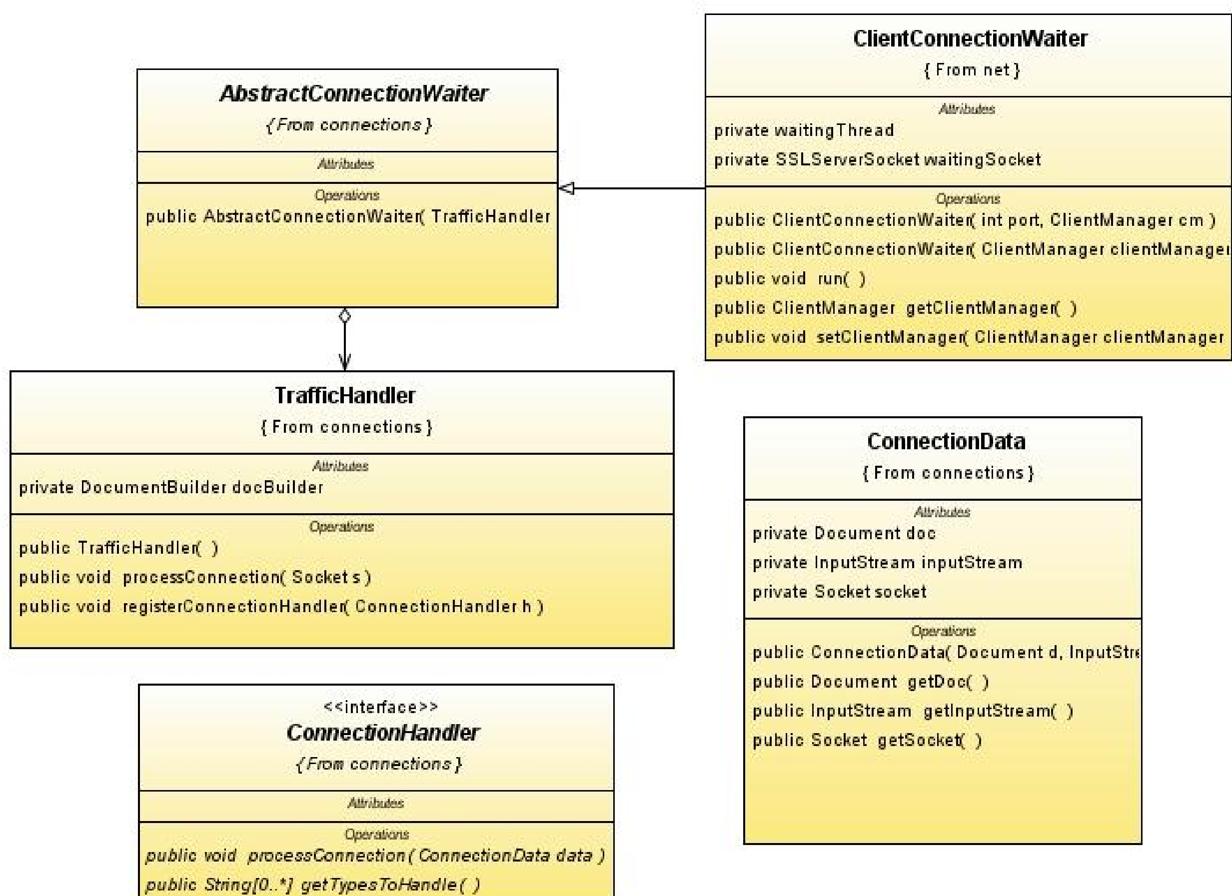
2.2.2.3. Komunikácia

Komunikačné protokoly

Pri návrhu komunikačných protokolov sme zvolili použitie jazyka XML, a to z viacerých dôvodov. Jednak sú ľahko spracovateľné - Java nám poskytuje na to silné nástroje, sú variabilné a ľahko rozšíriteľné. Konkrétne protokoly sme konštruovali pri implementácii. Pravidlá, ktoré sme dopredu určili, boli nasledovné:

- Koreňová značka každej správy má meno „msg“; výnimku tvoria len správy, ktoré sú poslané ako prvé po nadviazaní pripojenia, koreňové značky týchto správ sú nazvané „con“.
- Každá koreňová značka má atribút „type“, čo označuje typ správy. Tento atribút má veľmi dôležitú úlohu pri spracovaní správ, aj pri smerovaní správ na nasledovné spracovanie.

Pripojenia



Obr. 5: Diagram tried architektúry pripojenia

Dôležitou požiadavkou kladenou na sieťovú aplikáciu je, aby čo najmenej zaťažovala systémové nástroje. Pri sieťovej komunikácii je dôležité, aby sme obsadili čo najmenej portov. Jednou z ideí, na základe ktorej sme sa pri navrhovaní systému orientovali, bolo rozdelenie komunikácie - teda poskytovanie informácií o klientovi, respektíve o zdieľaných dátach - a prenášania súborov. Vzhľadom na to, že informácie o súboroch zdieľaných na klientskej strane nemusíme nutne získať od klienta, ale môže nám ich poskytovať napr. aj server, najprv sme si zvolili prístup dvoch počúvajúcich portov: jeden port slúži na poskytovanie súborov, a každý, kto si chce stiahnuť súbor, pripája sa na tento port, a pošle správu o žiadanom súbore. Na poskytovanie informácie o zdieľaných dátach slúži ďalší port. Neskôr sme zistili, že hoci táto idea znie dosť prirodzene, zbytočne by sme pri jej realizácii obsadili dva porty. Do architektúry sme pridali ďalšiu vrstvu, ktorá presmeruje pripojenie podľa

prvej správy, ktorú dostaneme. Presmerovanie môžeme veľmi jednoducho uskutočniť podľa atribútu „type". Táto architektúra je spoločná pre serverovú aj pre klientsku aplikáciu. Preto sme vytvorili jednu abstraktnú triedu AbstractConnectionWaiter, ktorá bude predkom všetkých tried, v ktorých počúvanie na porte je uskutočnené napr. postredníctvon ClientConnectioWaiter v klientskej aplikácii. Táto trieda má referenciu na jednu inštanciu triedy TrafficHandler, v ktorej je uskutočnené samotné presmerovanie. Každá trieda, ktorá spracováva pripojenie, implementuje ConnectionHandler interfejs, ktorý definuje typy prijateľných pripojení – tieto reťazce znakov sú totožné s reťazcami v atribúte "type" v prvej správe pri pripojení, a metódu processConnection(ConnectionData d) ktorá je zavolaná TrafficHandlerom v prípade, že prišlo pripojenie s vhodným typom. Následne sa odovzdajú ConnectionHandlerovi všetky informácie, ktoré by mohli byť potrebné pri ďalšom spracovaní pripojenia.

Táto architektúra má výhodu aj v tom, že XML dokumenty sú spracovávané na jednom mieste. Vďaka tomu triedu, ktorá túto úlohu uskutočňuje nemusíme inštanciovať viackrát, resp. nemusíme synchronizovať spracovávanie.

Komunikačný most

Veľkým problémom v peer-to-peer komunikácii je, že môže nastať situácia, kedy nebudeme vedieť uskutočniť priamu komunikáciu medzi dvoma počítačmi, v našom prípade medzi dvoma klientmi. Takáto situácia môže nastať z viacerých príčin. Môže ju spôsobiť firewall, alebo preklad sieťových adries (NAT), v tomto prípade klient nemá verejnú IP adresu a nie je možné sa na ňu pripojiť z inej siete ako z lokálnej – takého klienta budeme volať „pasívny" a opačnom prípade ako „aktívny". V niektorých prípadoch sa problém dá riešiť pomocou portforwardingu, ale to nám nedáva všeobecné riešenie.

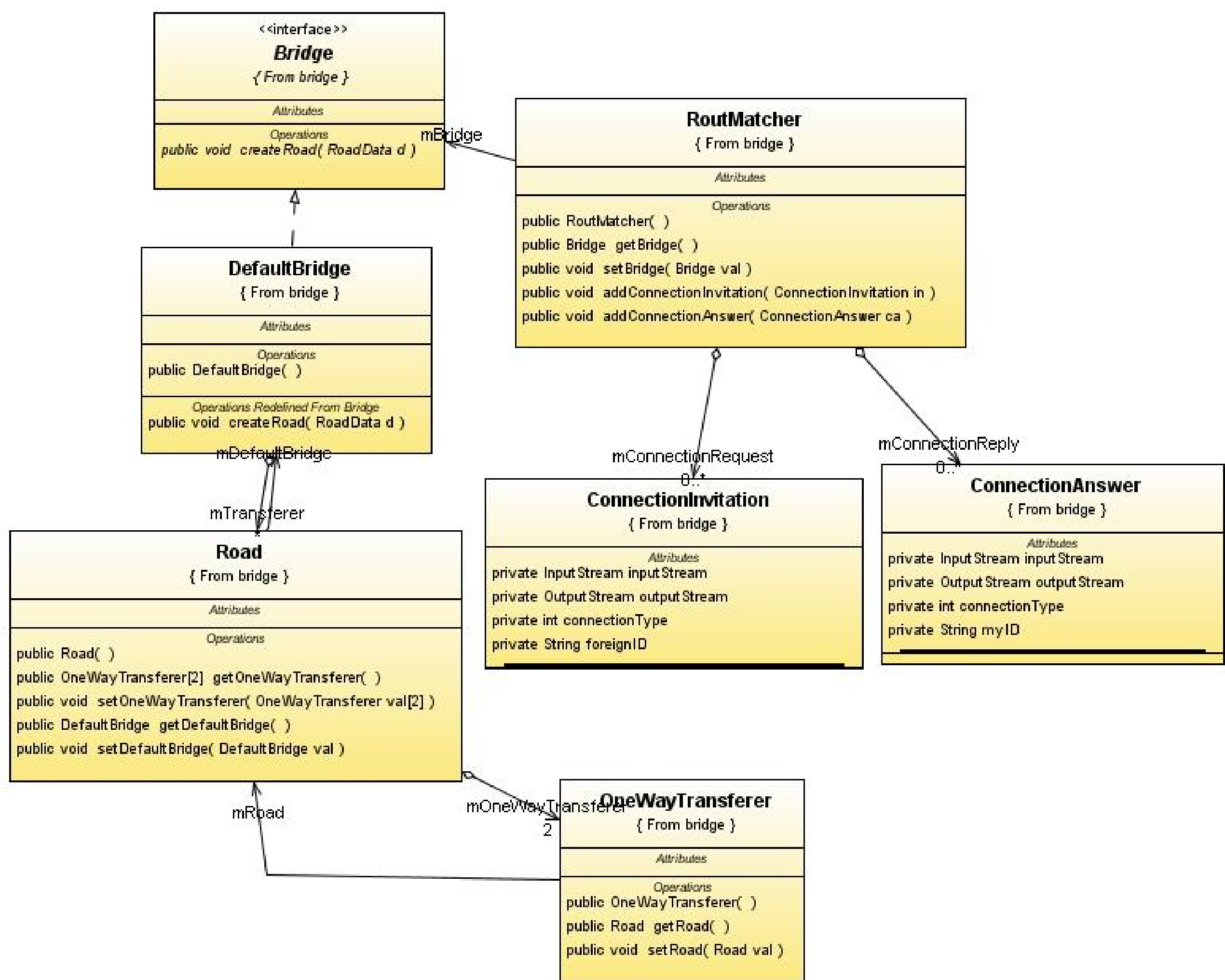
Situáciu môžeme riešiť celkom jednoducho, keď máme k dispozícii server a z dvoch klientov len jeden je pasívny. (Samozrejme aktívny klient sa chce pripojiť na

pasívneho.) V tomto prípade, keď obidvaja klienti sú pripojení na server, tak server môže žiadať na žiadosť aktívneho klienta, aby pasívny sa naňho pripojil, tým sa uskutoční pripojenie medzi aktívnym a pasívnym klientom na žiadosť aktívneho klienta, a máme problém vyriešený.

Oveľa ťažšie riešiteľným prípadom je, keď chceme nadviazať spojenie medzi dvoma pasívnymi klientmi. Na riešenie tohto prípadu sme vytvorili časť systému, ktorú sme nazvali komunikačný most.

Komunikačný most sme navrhovali úplne nezávisle od klientskej aj serverovej časti systému. Most funguje ako server, ktorý prenáša dáta medzi dvoma klientmi. Keď sa klient pripája na most, poskytuje informácie o tom, o komunikáciu akého typu ide, resp. akú má úlohu pri komunikácii. Uvedie, či cieľom pripojenia je prenášanie súboru, alebo len jednoduchá výmena informácií medzi klientmi, ďalej informuje most o tom, či je žiadateľom alebo je žiadaný o komunikáciu. V prípade, že sa naňho pripojí ďalší klient s vyhovujúcimi vlastnosťami, most vytvorí spojenie medzi nimi a bude slúžiť ako sprostredkovateľ dát, teda pošle ďalej všetky dáta, ktoré dostal od prvého klienta k druhému a opačne.

Most bol separovaný od serverovej aplikácie hlavne kvôli všeobecnosti čo sa týka rozširovateľnosti. Hoci pri špecifikácii bolo dohodnuté, že most bude konkrétne viazaný k serveru, bude môcť stáť aj úplne samostatne na inom počítači, alebo môže byť aj časťou klientskej aplikácie. V závislosti od tohto faktu sme navrhovali aj komunikačné protokoly týkajúce sa mostu, aj architektúry tried.



Obr. 6: Diagram tried pre komunikačný most.

S interfejsom Bridge definujeme všetky funkcie, ktoré by mal most implementovať. Konkrétna implementácia je trieda DefaultBridge, ktorá implementuje jedinú metódu Bridge interfejsu: createRoad(RoadData d). RoadData obsahuje všetky potrebné informácie na prenášanie dát. Trieda Road reprezentuje obojsmernú komunikáciu, tvoria ju dve inštancie triedy OneWayTransferer, ktorá realizuje jednosmernú komunikáciu. Prenášanie dát je nezávislé na type, vo všeobecnosti Bridge nevie nič o tom, čo prenáša, len si prečíta istý počet bajtov z jednej strany a prenáša ich na druhú.

Trieda RoutMatcher slúži na nájdenie dvoch k sebe prislúchajúcich klientov. Podľa

toho, či je klient žiadateľom pripojenia alebo žiadaný, vytvoria sa triedy ConnectionInvitation a ConnectionAnswer, ktoré obsahujú informácie o tom, akého klienta hľadajú. Klienti sa nájdu podľa identifikátora žiadaného. Museli sme rozlíšiť aj typ komunikácie, a to kvôli implementácii, keďže už vytvárame triedy, ktoré realizujú komunikáciu predtým, ako je pripojenie uskutočnené.

2.2.3. Implementácia

Fáza implementácie nastane, keď sa už vytvára samotné programové dielo. Na začiatku tejto etapy zvolíme programovací jazyk, v ktorom systém budeme implementovať. Vo všeobecnosti toto rozhodnutie závisí od viacerých faktorov, ako sú skúsenosť, úroveň abstrakcie, podporné nástroje. V našom prípade výber programovacieho jazyka bol jednoznačný, ako sme to uviedli už pri návrhu systému. Za programovací jazyk sme zvolili programovací jazyk Java, konkrétne Java SE 1.6.

Po zvolení programovacieho jazyka nastupuje fáza tvorby. Ešte predtým, ako sa pustíme do tvorby systému, je dobré stanoviť pravidlá písania samotného kódu:

- Štandardné pomenovanie premenných, konštánt, metód a tried.
- Štandardné formátovanie textu. (Hoci v súčasnosti formátovanie textu už nie je ozajstný problém z dôvodu, že moderné editory to spravia za nás.)
- Dokumentácia kódu. Dokumentácia kódu má byť abstraktná, teda nezávislá na implementácii, písaná v prirodzenom jazyku. Dokumentovať by sme mali všetky verejné a chránené metódy triedy. V našom prípade Java poskytuje silný nástroj na dokumentáciu menom JavaDoc. Okrem dokumentácie kódu by sme mali zvlášť dokumentovať abstraktnejšie funkčnosť celého systému.
- Vyhnutie sa programátorským trikom – kód sa stane ťažko čitateľným a nezrozumiteľným, ťažko pochopiteľným.

V našom prípade písania budeme dodržiavať štandardy všeobecne používané v štandardnej knižnici. (Napr. triedy sa začínajú veľkými písmenami, metódy a premenné malými atď.)

2.2.3.1. Výber vývojového prostredia

Pred písaním kódu sme tiež museli vybrať vývojové prostredie. Naša voľba padla na NetBeans IDE namiesto Eclipse kvôli možnosti jednoduchého vytvárania grafického užívateľského prostredia.

2.2.3.2. Databázy

Pri implementácii našou úlohou bolo vybratie vhodnej databázy pre našu aplikáciu. Keďže ide o multiplatformovú aplikáciu, ktorá má byť ľahko prenášateľná, rozhodli sme sa pre čisto javovskú databázu menom Derby. Derby možno použiť ako embedded databázu, teda databázu, ktorá nepotrebuje, aby bežal server: všetky operácie vykoná priamo na pevný disk, čo nám veľmi vyhovuje, lebo nemusíme inštalovať veľký databázový systém pri inštalácii softvéru.

Derby umožní vytvorenie šifrovanej databázy, čiže všetky dáta, ktoré uloží na pevný disk, budú šifrované. K pripojeniu sa na databázu potrebujeme heslo, takže bez hesla z nej nie je možné čítať.

Poskytuje aj ďalšiu funkcionalitu; v prípade, že všetky operácie vykonáme cez jediné pripojenie, nemusíme sa zaoberať synchronizáciou, spraví to za nás. V našom prípade je to realizované tak, že pri spustení aplikácie vytvoríme spojenie s databázou, a toto spojenie, ktoré v našom prípade je inštancia triedy Connection, poskytujeme každej triede, ktorá ho potrebuje.

V Jave komunikácia s databázami prebieha cez JDBC API. Tento API nám poskytuje všeobecný interfejs, cez ktorý pristupujeme k databáze. Poskytuje množinu abstraktných tried a interfejsov, ktoré databáza alebo prístupový „most" k databáze musí implementovať. JDBC API nám poskytuje triedu, ktorá umožňuje prekompiláciu SQL dotazov alebo operácií menom PreparedStatement. Je to veľmi

užitočné, lebo urýchľuje vykonávanie operácií nad databázou.

2.2.3.3. Bezpečnosť a komunikácia

V špecifikácii sme stanovili, že jednou z funkcionalít, ktoré náš systém musí implementovať, bude bezpečná komunikácia. Keďže našimi komunikačnými kanálmi sú počítačové siete, mali sme dve možnosti:

- Implementovať vlastné protokoly nad štandardnou sieťovou komunikáciou. Vytvoriť nešifrované spojenie, a pomocou Java Cryptography Extension si vytvoriť naše vlastné protokoly.

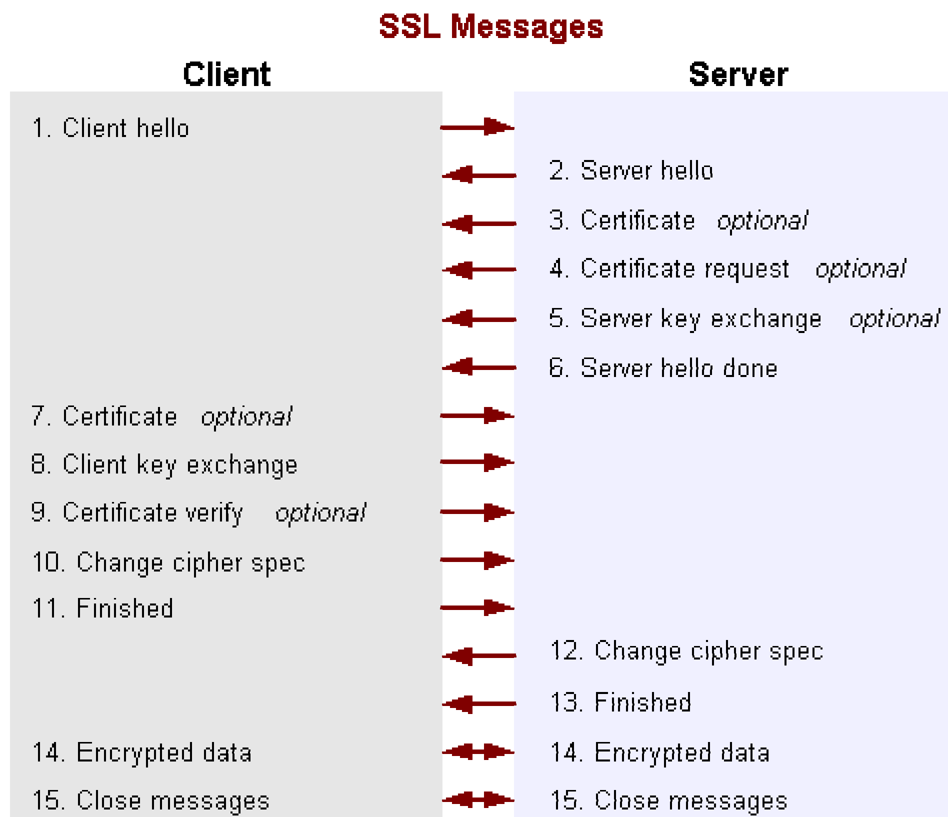
- Používať JSSE – Java Secure Socket Extension, čo je štandardná knižnica implementujúca SSL/TLS protokoly.

Aj z pohľadu bezpečnosti, aj kvôli náročnosti implementácie vlastných protokolov sme sa jednoznačne rozhodli pre použitie JSSE a SSL protokolov. SSL je protokol, ktorý nám poskytuje ďalšiu bezpečnú vrstvu, prenosnú vrstvu a aplikačnú vrstvu:

TCP/IP Layer	Protocol
Application Layer	HTTP, NNTP, Telnet, FTP, etc.
Secure Sockets Layer	SSL
Transport Layer	TCP
Internet Layer	IP

Obr. 7: Umiestnenie Secure Socket Layera.

Tým umožňuje transparentnú bezpečnú komunikáciu. SSL protokol v našom prípade prebieha nasledovne: po pripojení sa na server ho klient žiada o autentifikáciu, autentifikácia prebieha pomocou elektronických certifikátov. Keď klient overil, že server má platný certifikát, dohodnú sa na súkromnom kľúči, pomocou ktorého budú prenášané dáta šifrované. Ilustrácia kompletného protokolu je na Obr. 8.



Obr. 8: SSL protokol

V našom prípade sme použili self signed certifikáty, Java JDK nám poskytuje program, ktorý umožňuje generovať certifikáty, a bezpečne ich ukladať. Na ukladanie slúžia takzvané KeyStore a TrustStore:

- KeyStore – súbor, ktorý obsahuje v sebe samotný certifikát aj so súkromným kľúčom, tiež v šifrovanej podobe. Tento súbor je potrebný pre server, aby sa vedel preukázať klientovi.

- TrustStore – súbor, ktorý obsahuje v sebe informácie o dôveryhodných certifikátoch, je používaný klientom.

Použitím týchto súborov je možné vytvoriť inštancie SSLSocket a SSLServerSocket tried, cez ktoré už môže prebiehať bezpečná obojsmerná komunikácia, prenos dát.

2.2.3.4. Komunikačné XML protokoly

Všetky komunikácie medzi klientmi, resp. klientom a serverom prebiehajú cez XML dokumenty. Na vytváranie správ nepoužívame žiadne špeciálne API. Vzhľadom na to, že sú to väčšinou dosť jednoduché dokumenty, to ani nepotrebujeme. Na zostrojenie týchto reťazcov používame triedu StringBuilder, čím dynamicky vieme rozširovať jednotlivé reťazce bez toho, aby vytvorili nové objekty.

Pri vytváraní XML správ si musíme dávať pozor na dve veci:

- V jazyku XML je definovaných 5 špeciálnych znakov, ktoré pri konštrukcii dokumentu musíme nahradiť špeciálnymi reťazcami. Napríklad znak „&“ musíme nahradiť reťazcom „&“. Na toto nám slúži vlastná trieda menom *XmlSpecChConv*, ktorá má jedinú statickú metódu *convert(String)*, ktorý z reťazca vytvorí pretransformovaný výsledok, už bez špeciálnych znakov.
- Reťazec nemôže obsahovať znak reprezentujúci nový riadok. Táto podmienka je dôležitá pri spracovávaní správy. Hoci nikde pri protokoloch sme ani nepotrebovali písať takéto znaky, ale v prípade potreby by sme sa s nimi mohli zaobchádzať podobne, ako so špeciálnymi znakmi, vymenili by sme ich za iné, a neskôr by sme ich transformovali naspäť.

Spracovávanie XML správ prebieha spôsobom, že počkáme, kým sa neprečíta celý XML dokument, celú správu parsujeme pomocou triedy DocumentBuilder, ktorý nám vráti správu vo forme inštancie triedy Document, a z tejto inštancie vyčítame všetky potrebné informácie. Na to, aby sme si prečítali celý dokument, slúži špeciálny znak na konci dokumentu, v našom prípade znak nového riadku. Na čítanie

takýmto spôsobom nám Java poskytuje triedu BufferedReader.

Príklady protokolov

```
<con type="toserver">
  <name>MyClientsName</name>
  <description>mail: fakemail@gmail.com</description>
  <ispassive>false</ispassive>
  <shareamount>1233344</shareamount>
  <conport>4678</conport>
</con>
```

Obr. 9: Príklad reálneho XML protokolu pri pripojení na server.

```
<msg type="reqservertopassive-com">
  <id>224678690</id>
</msg>
```

Obr. 10: Príklad XML protokolu, klient žiada server, aby sprostredkoval spojenie medzi aktívnym a pasívnym klientom

2.2.3.5. Hľadanie klientov na lokálnej sieti

Jednou z funkcionalít klientskej aplikácie je hľadanie možných klientov na lokálnej sieti. Na túto funkcionalitu tiež platí požiadavka, aby bola nezávislá od operačného systému.

Implementácia je podstate veľmi jednoduchá. Je založená na protokoloch UDP, ktoré nám poskytujú možnosť vysielania paketov na lokálnej sieti. Klientska aplikácia automaticky čaká na paket takého typu, a pošle správu naspäť so všetkými informáciami o klientovi, ktoré sú potrebné na to, aby s ním mohlo byť uskutočnené spojenie, ďalej také základné informácie, ako meno a krátky popis klienta.

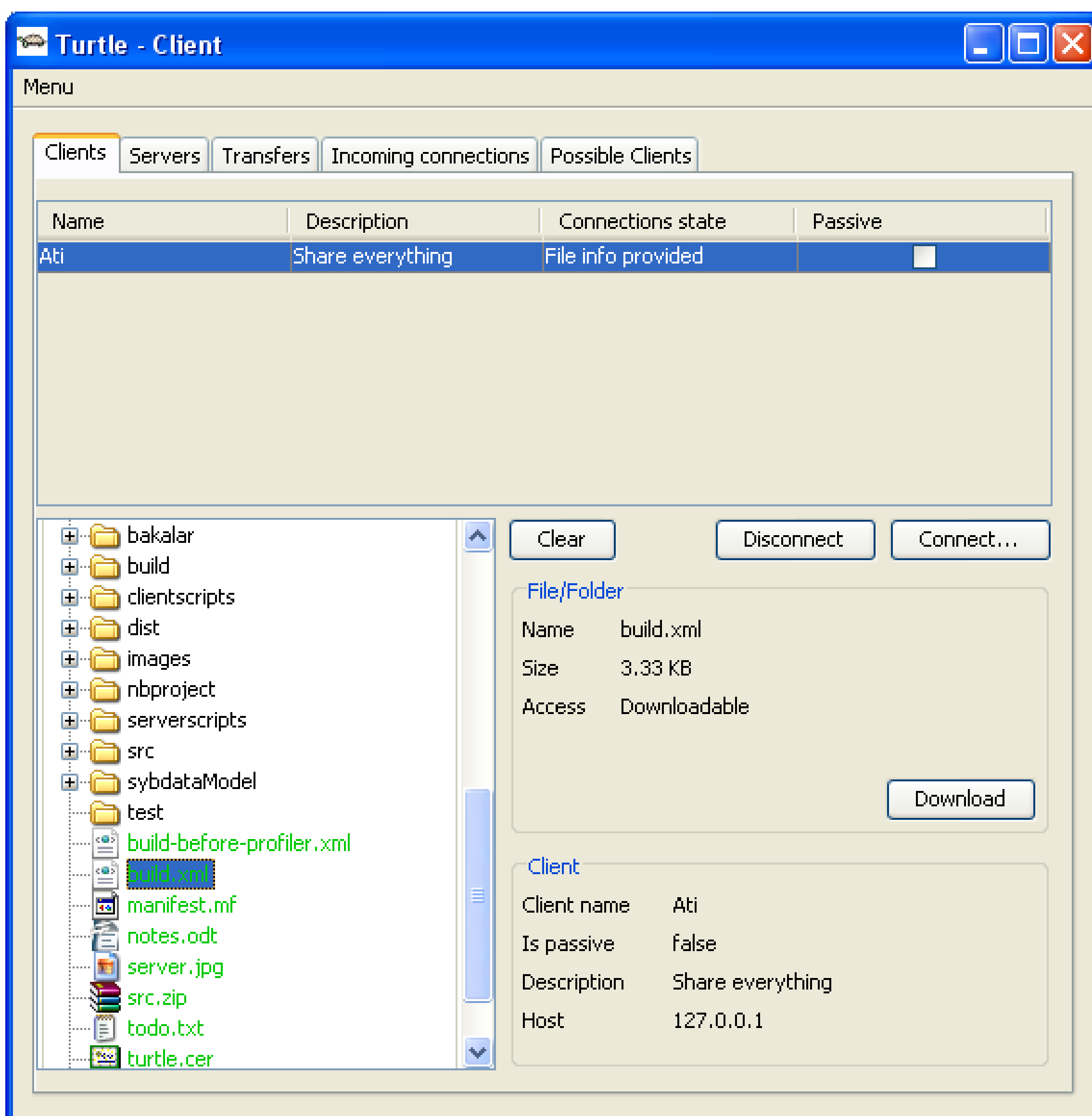
2.2.3.6. Grafické používateľské rozhranie

Vytváranie grafického používateľského rozhrania je vo všeobecnosti náročná a komplexná úloha. Pri vytváraní tejto časti aplikácie musíme dávať pozor na niekoľko požiadaviek. Jednak grafické používateľské rozhranie musí odzrkadľovať funkcionalitu softvéru, musí byť ľahko použiteľné, prehľadné a zrozumiteľné. Užívateľ sa v ňom musí vedieť okamžite zorientovať, resp. čas venovaný na jeho naštudovanie musí byť minimálny. Okrem toho moderné používateľské grafické rozhrania sledujú aj isté dominantné trendy. Hoci v našom systéme grafické používateľské rozhranie nebolo prioritou, snažili sme sa dodržať požadované koncepcie.

V Jave paleta možností pri vytváraní grafických používateľských rozhraní je veľmi široká. Jednak máme k dispozícii rôzne grafické knižnice ako SWT, Swing alebo AWT. Okrem toho v súčasnosti už existujú aj veľmi inteligentné editory, ktoré nám veľmi uľahčujú prácu s týmito knižnicami a grafickými komponentmi. Naša voľba padla na Swing, a to z viacerých dôvodov. Jednak je to veľmi vyvinutá a asi aj najrozšírenejšia grafická knižnica, je časťou štandardnej javovskej knižnice, jednak existujú veľmi silné prostriedky, pomocou ktorých sa s ňou dá ľahko a efektívne pracovať. Pri vytváraní nášho grafického používateľského rozhrania sme používali editor s názvom Matisse, ktorý je integrovaný do nášho vývojového prostredia.

Popíšeme len hlavné koncepcie, ktorými sme sa riadili pri dizajne. Pri vytvorení nášho grafického používateľského rozhrania bolo hlavným cieľom rozdeliť funkcionality do jednotlivých skupín pomocou záložiek. Takže naším centrálnym komponentom je JTabbedPane, ktorý implementuje túto funkcionalitu. V jednotlivých záložkách sú vytvorené ďalšie komponenty, ktoré sú prostriedkami na kontrolovanie istej funkcionality. S istými vedľajšími funkcionalitami, ktoré sú zriedkavejšie používané, sme zaobchádzali inak. Pre ne sme vytvárali nové dialógové

okná, ktoré sú dostupné z hlavného menu.



Obr. 11: Grafické používateľské rozhranie

2.2.4. Možnosti v budúcnosti

Pojem „hotový“ v softvérovom inžinierstve neexistuje, softvér alebo softvérový systém sa dá vždy vylepšovať, optimalizovať, rozšíriť novými funkcionalitami. Platí to aj pre náš softvérový systém, pri implementácii ktorého nám napadli nové a nové možnosti. Okrem toho existujú podobné systémy, v ktorých sú

funkcionality implementovateľné aj v našom systéme. Práve preto sme sa snažili vytvoriť našu architektúru tak, aby bola ľahko rozšíriteľná. Tým sme nechali otvorenú bránu do budúcnosti pre potenciálnych vývojárov.

2.2.5. Záver

Spracovanie bakalárskej práce bola výborná skúsenosť, ktorá mi umožnila vyskúšať si v praxi poznatky nadobudnuté počas štúdií na FMFI UK. Mal som možnosť aplikovať svoje poznatky hlavne z oblasti programovania, z počítačových sietí a softvérového inžinierstva.

Dúfam, že moja práca si nájde svoje miesto v reálnom živote.

Vďaka tejto práci som získal množstvo skúseností, ktoré sú devízou do budúceho profesionálneho života.

3. Použitá literatúra

1) The Java Tutorial
   http://java.sun.com/docs/books/tutorial/
2) Mederly Pavol
   Princípy tvorby softvéru, materiál k prednáškam
   FMFI UK
3) Scott Oaks, Henry Wong
   Java Threads, 3rd Edition
4) Elliotte Rusty Harold
   Java Network Programing, 3rd Edition
5) Wikipedia, the free encyclopedia
   http://www.wikipedia.org

4. Prílohy

1) Popis XML protokolov
   Ako prílohu v elektronickej podobe poskytujeme popis XML protokolov v jazyku DTD. (umiestenie na cd: \XmlProtocolsDTD.pdf)
2) Dokumentáciu vo forme JavaDoc.
   Ako prílohu v elektronickej podobe poskytujeme dokumentáciu vo forme JavaDoc.(umiestenie: \javadoc\)
3) Zdrojový kód aplikácie, skripty.
   Tiež v elektronickej podobe poskytujeme zdrojové kódy aplikácie, a SQL skripty na vytvorenie databázy. (umiestnenie zdrojového kódu: \src\;

umiestnenie skrípt: \scripts\ )

4) Návod na použitie

Systém pre zdieľanie súborov v počítačových sieťach - Návod na použitie

1. Úvod

Systém slúži na zdieľanie súborov v počítačových sieťach, je založený na tzv. peer to peer technológii. Hlavnou časťou systému je klientska aplikácia, ktorá bude centrálnou témou tohto návodu. Návod k serverovej časti tu neuvádzame kvôli jednoduchosti tejto časti systému.

2. Inštalácia

Na prevádzku systému je potrebná Java Runtime Enviroment, verzia 1.6 a vyššie. Systém je nezávislý od operačného systému. Stačí si ho kopírovať na príslušné miesto a aplikáciu spustíme pomocou *turtle.jar*.

3. Používanie systému

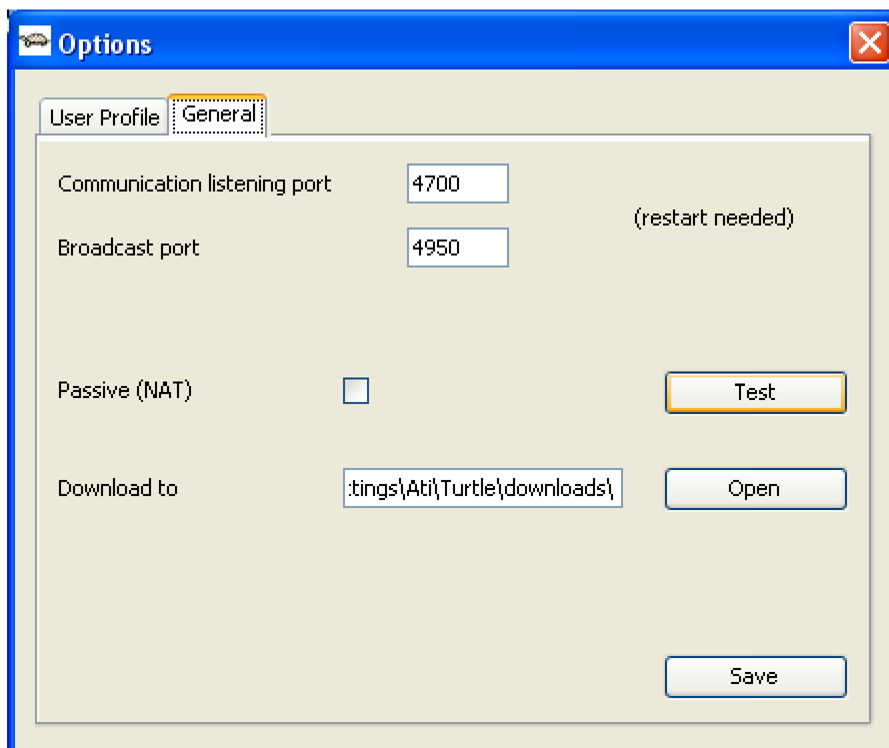
Klientska aplikácia poskytuje grafické používateľské rozhranie, cez ktoré sa ľahko a jednoducho dá manipulovať s funkciami systému. Komponenty grafického používateľského rozhrania sú rozdelené do dvoch kategórií: na komponenty, ktoré riadia frekventovane používané funkcionality, a na ostatné menej používané. Komponenty druhej skupiny sú prístupné cez hlavné menu, napr. Options, Share Manager. Komponenty prvej skupiny sú podľa skupín rozdelené do záložiek.

4. Nastavenie systému

K nastaveniam systému sa dostaneme cez hlavné menu (Menu -> Options). Tu môžeme nastaviť profil používateľa, jeho meno, heslo – pozri prístupové práva, a popis používateľa. Popis je krátka správa o klientovi.

Ďalšie nastavenia na prevádzku systému:

- *Communication listening port* – port, cez ktorý sa očakáva pripojenie ďalšieho klienta, pri zmene musí byť aplikácia reštartovaná.
- *Broadcast port* – port, ktorý slúži na hľadanie klientov na lokálnej sieti, na tomto porte čakáme žiadosť o poskytovanie informácií od nášho klienta, tiež odtiaľ rozposielame žiadosti o informácie o klientoch na lokálnej sieti. Pri zmene musí byť aplikácia reštartovaná.
- *Passive* – veľmi dôležitá vlastnosť systému, znázorňuje, či máme verejnú IP adresu. Musí byť označené, keď nemáme verejnú IP adresu. Systém ponúka automatický test na zistenie tohto atribútu.
- *Download to* – miesto, kam sú stiahnuté súbory.



Obr. 12: Nastavenie systému

5. Bezpečnosť

Sieťová komunikácia

V systéme sú všetky komunikácie šifrované pomocou používania SSL a vlastných certifikátov. Výnimkou je len rozposielanie paketov za účelom hľadania klientov na lokálnej sieti.

Rozšírenia pri bezpečnosti

V prípade, že užívateľ chce používať iné certifikáty, než aké sú poskytované systémom, môže si ich vymeniť. Použité certifikáty sú ukladané do súborov:

- clientscripts/truststr.jks – všetky dôveryhodné certifikáty.
- clientscripts/keystr.jks – súbor na ukladanie certifikátu, ktorý je použitý pri vytváraní pripojenia.

Na manipulovanie s týmito súbormi, prípadne na vytvorenie nových, poskytujeme program *clientscripst/keytool.exe.* Podrobný popis tohto softvéru je popísaný na stránke:

http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html

6. Zdieľanie

Prístupové práva

Každý zdieľaný súbor a adresár v našom systéme má svoje prístupové práva. Môžeme určiť všeobecné prístupové práva, čo sú takzvané minimálne

prístupové práva, ktoré platia pre všetkých klientov pripojených na nás (ktorí nie sú registrovaní).

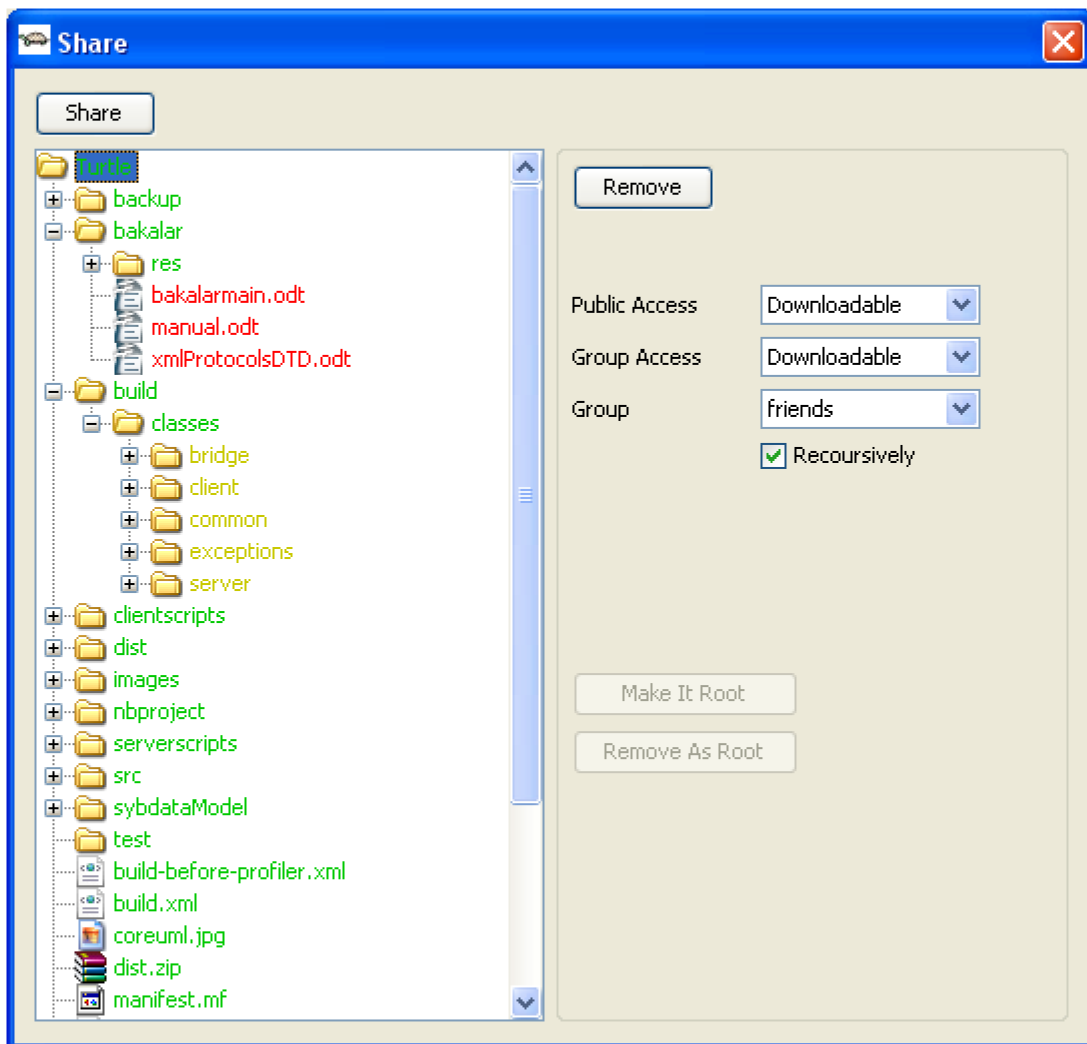
Okrem toho každý súbor a adresár patrí do nejakej skupiny, môžeme určiť prístupové práva súboru pre ľudí, ktorí sú zaregistrovaní do tejto skupiny. (Keďže klient, ktorému poskytujeme informácie o zdieľanom súbore je zaregistrovaný do skupiny do ktorej patrí aj súbor, dostane maximum zo všeobecného prístupového práva a z prístupového práva skupiny. Napríklad keď všeobecné prístupové právo súboru je „viditeľný“ a právo pre skupinu je „neviditeľný“, tak prístupové právo klienta bude „viditeľný“).

Prístupové právo k súboru a k adresáru môže byť:

- Not Visible – neviditeľný pre užívateľa. O týchto súborov nie sú poskytnuté žiadne informácie. Má význam napr. v situácii, keď nechceme poskytovať informácie o súbore všeobecne, ale len pre istú skupinu ľudí.
- Visible – viditeľný. Poskytujeme informácie o súbore, ale klient si ho od nás nemôže stiahnuť.
- Downloadable – stiahnuteľný. Poskytujeme informácie o súbore, a naviac klient si ho od nás môže stiahnuť.

Zdieľanie súborov

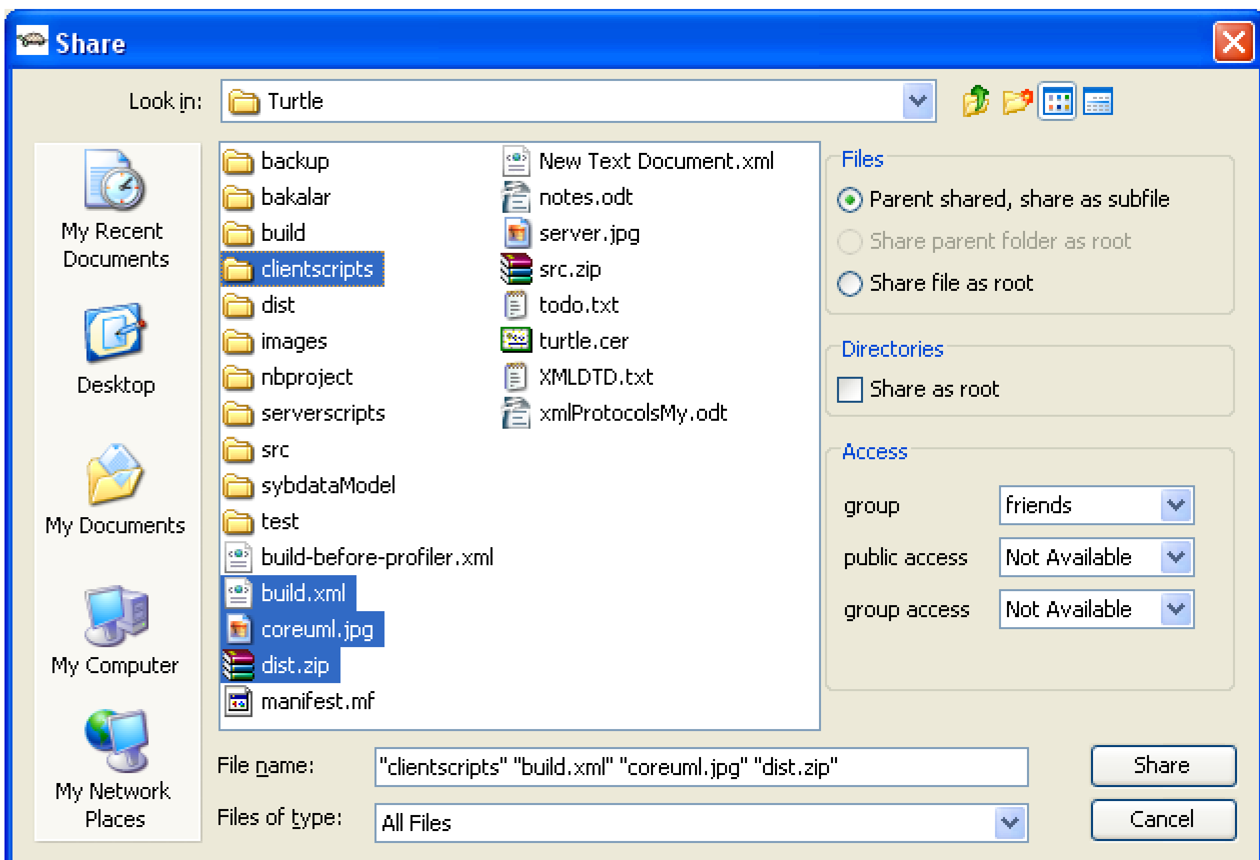
Zdieľanie súborov robíme pomocou tzv. Share Manager -a (Menu -> Share..).



Obr. 13: Share Manager

Keď klikneme na tlačítko „Share“, otvorí sa okno, kde môžeme označovať jednotlivé súbory a adresáre, ktoré budeme zdieľať. Už tu môžeme vybrať prístupové práva a skupinu, ktoré budú pre súbory platné. Pre vybraté adresáre a súbory môžeme nastaviť, či v strome zdieľaných súborov budú na prvej úrovni, teda že nebudú pod žiadnym adresárom. Keď sme už vybrali všetky súbory, ktoré chceme zdieľať, môžeme ďalej manipulovať so zdieľanými súbormi (Obr. 2: Share Manager). Tu môžeme podrobnejšie určiť prístupové práva a štruktúru zdieľaných súborov. Označením tlačítka „recoursively“ aplikujeme zmeny adresára na všetky podadresáre

a súbory pod ním.



Obr. 14: Výber súborov na zdieľanie

Registrácia užívateľov

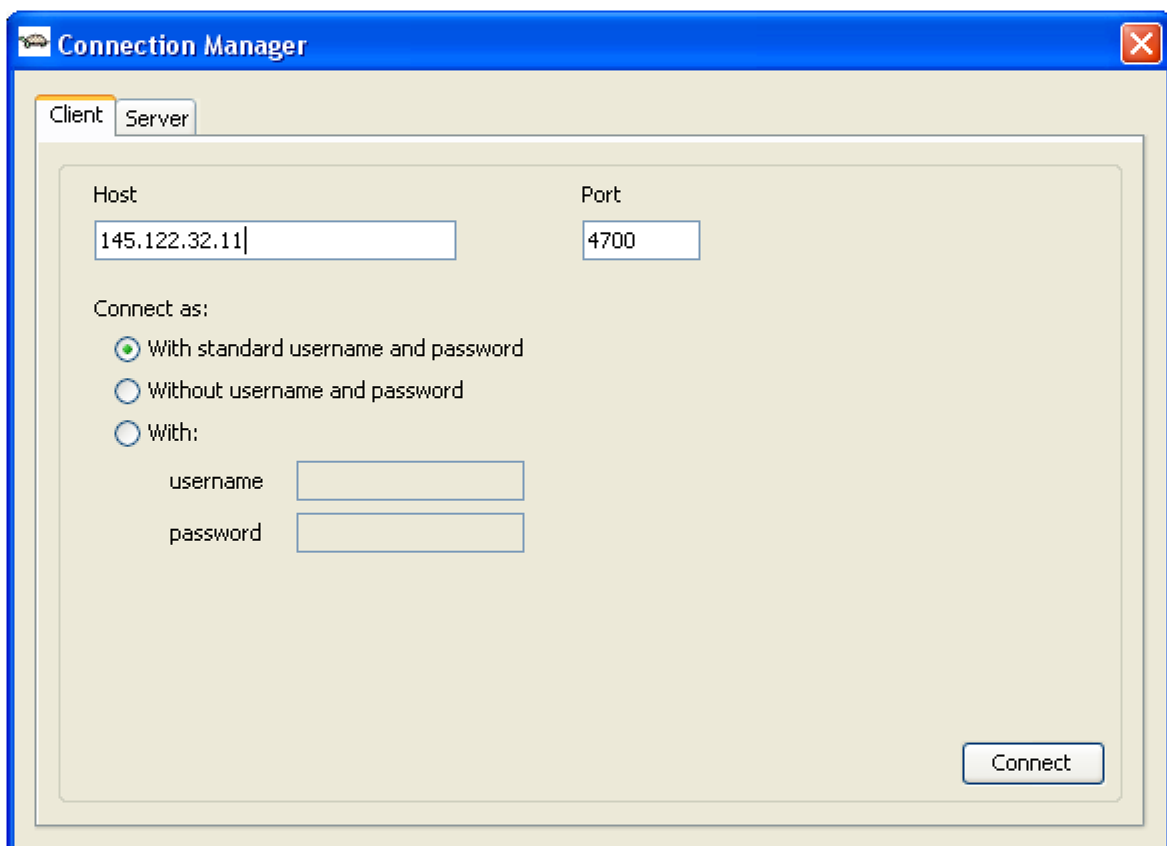
Na registráciu klientov slúži User Manager (Menu -> User Manager), tu môžeme zaregistrovať nových klientov, pridať nové skupiny a určiť, ktorý klient do akých skupín patrí. Túto časť kvôli jej jednoduchosti podrobnejšie neopisujeme.

7. Pripojenia na ďalšieho klienta, stiahnutie súborov

Na pripojenie sa na klienta, servera slúži Connection Manager. Tento

komponent systému je dostupný aj z hlavného menu, aj z jednotlivých záložiek, ktoré sú s ním logicky spojené (Clients, Servers). Pri pripojení sa na klienta musíme nastaviť jeho IP adresu a port, cez ktoré komunikuje. Pri pripojení máme na výber, s akým profilom sa pripájame:

- štandardným – definovaný pri nastaveniach,
- bez mena a hesla,
- s menom a heslom špecifikovanými na mieste.



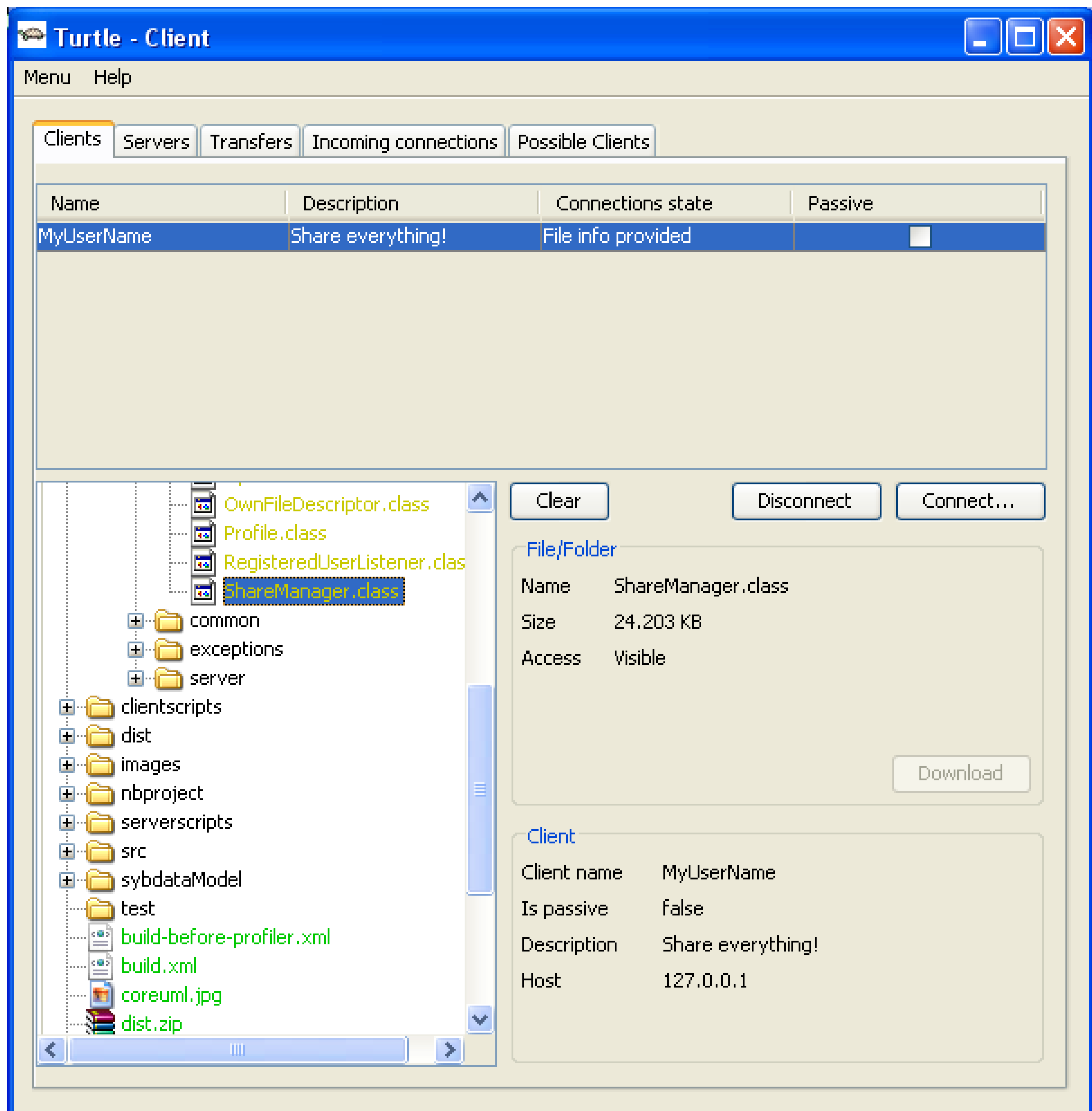
Obr. 15: Connection Manager

Keď sme sa úspešne pripojili na klienta, tak sa to objaví na záložke Clients v zozname klientov.

Každý klient má svoj stav (state):

- Initializing – stav inicializácie, ešte nie sú poskytnuté všetky informácie o klientovi.
- File info provided – všetky informácie sú poskytnuté.
- Disconnected – sme odpojení od klienta.

Keď si zo zoznamu vyberieme klienta, tak sa objavia zdieľané adresy a súbory.

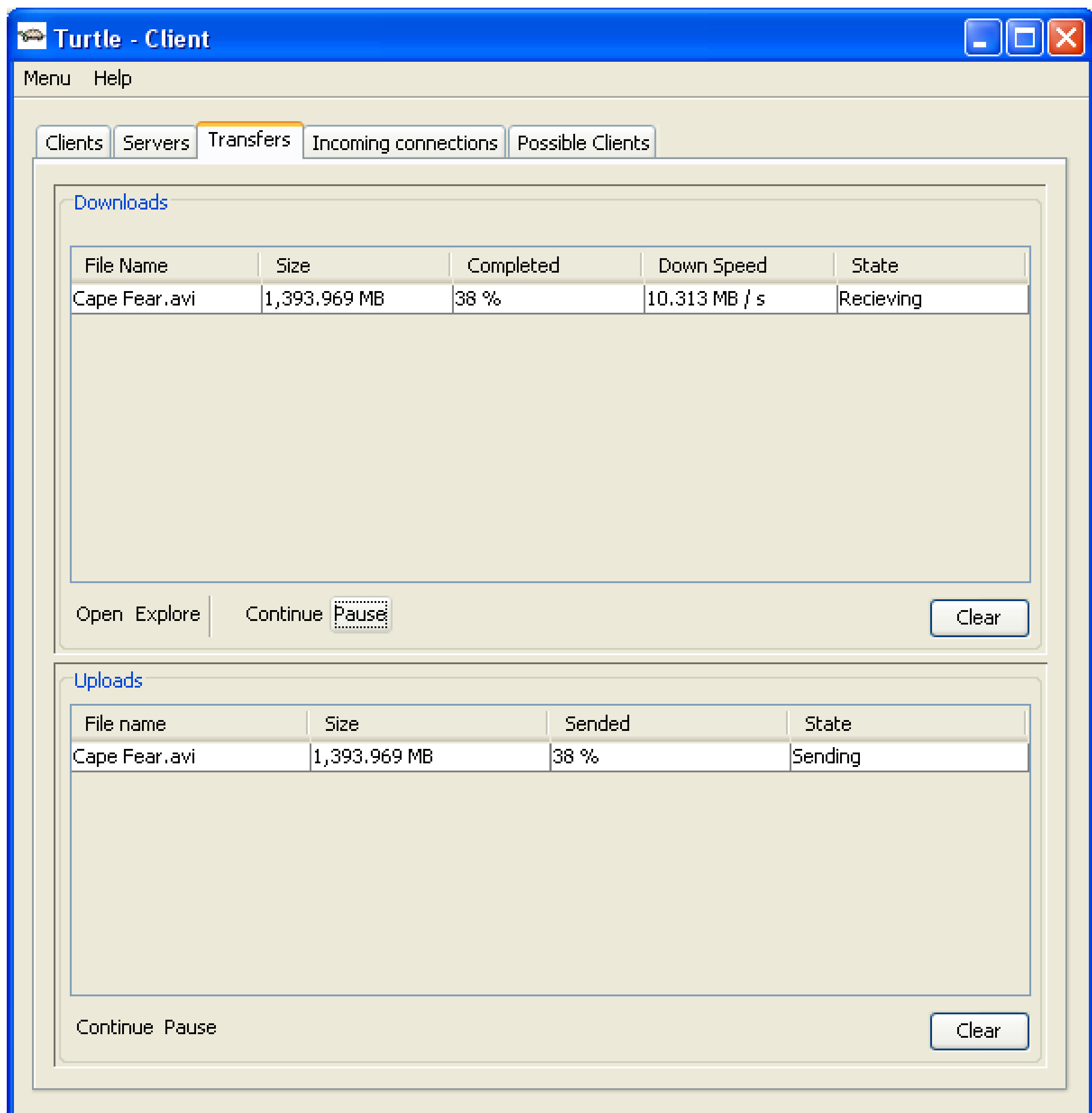


Obr. 16: Pripojené na klienta

Tu si môžeme vyberať, ktoré súbory si chceme stiahnuť, keď na to máme prístupové práva. Môžeme sa odpojiť od klienta (tlačítko Disconnect).

Vyčistenie klientov (tlačítko Clear) slúži na vyhodenie zo zoznamu tých klientov, od ktorých sme sa už odpojili.

Procesy stiahnutia súborov môžeme sledovať pod záložkou Transfers. V tomto prípade tlačítko Clear vyhodí tie položky, ktoré reprezentujú dokončené sťahovania.



Obr. 17: Stiahnutie a posielanie súborov

Stavy pri stiahnutí, resp. posielaní súborov (v nasledujúcom pod pojmom proces chápeme sťahovanie alebo posielanie súborov):

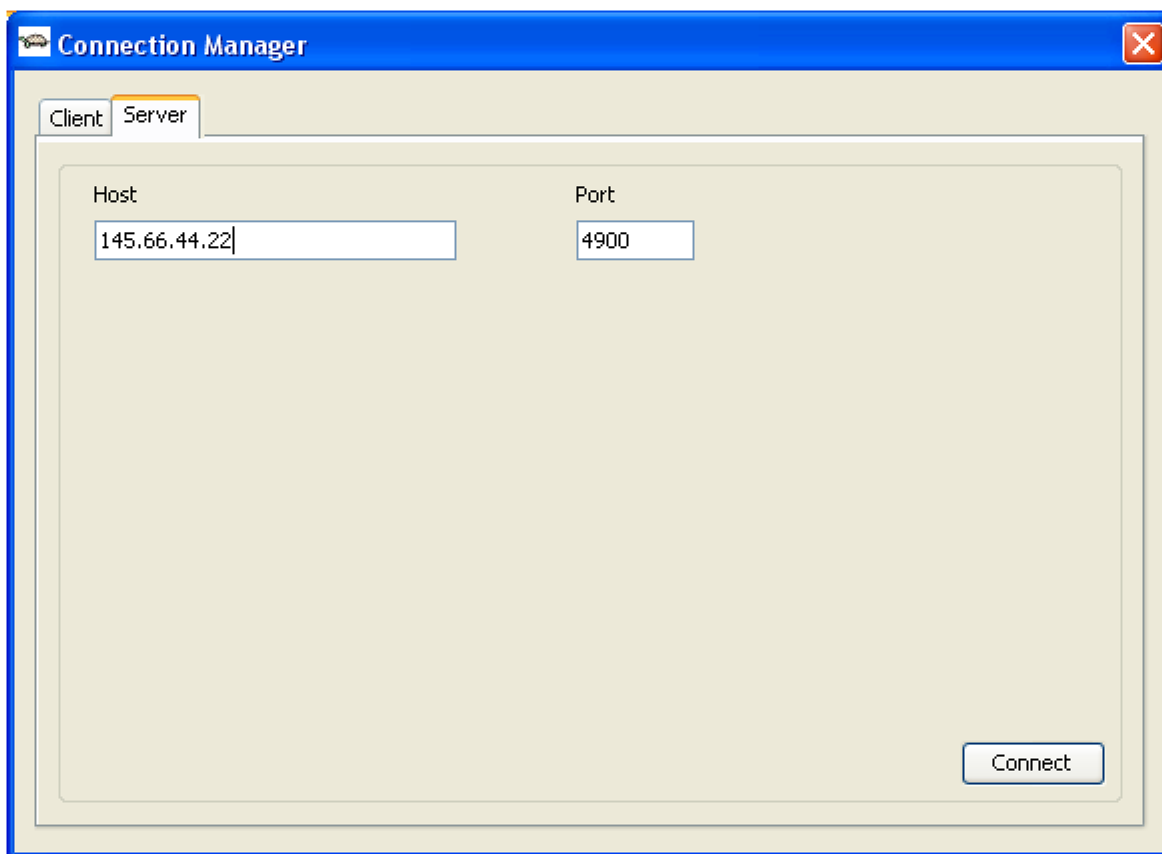
- Initializing – stav inicializácie.
- Recieving / Sending – stav, keď sa súbor práve sťahuje, resp. posiela.
- Recieve Completed / Send Completed – proces bol úspešne dokončený.
- Cannot start – stav nastane, keď z nejakého dôvodu proces nemohol byť spustený.
- Interrupted – stav nastane, keď proces bol z neznámeho dôvodu zastavený.
- Paused – proces bol dočasne zastavený (tlačítka Pause/Continue).

8. Pripojenie na server, pripojenia na klientov cez server

Server slúži na poskytovanie informácií o ďalších klientoch a na riešenie problémov vyskytujúcich sa pri pripojeniach v prípade neverejných IP adries (NAT, v nasledujúcom klienta s verejnou IP adresou budeme nazývať „aktívny“, v opačnom prípade „pasívny“):

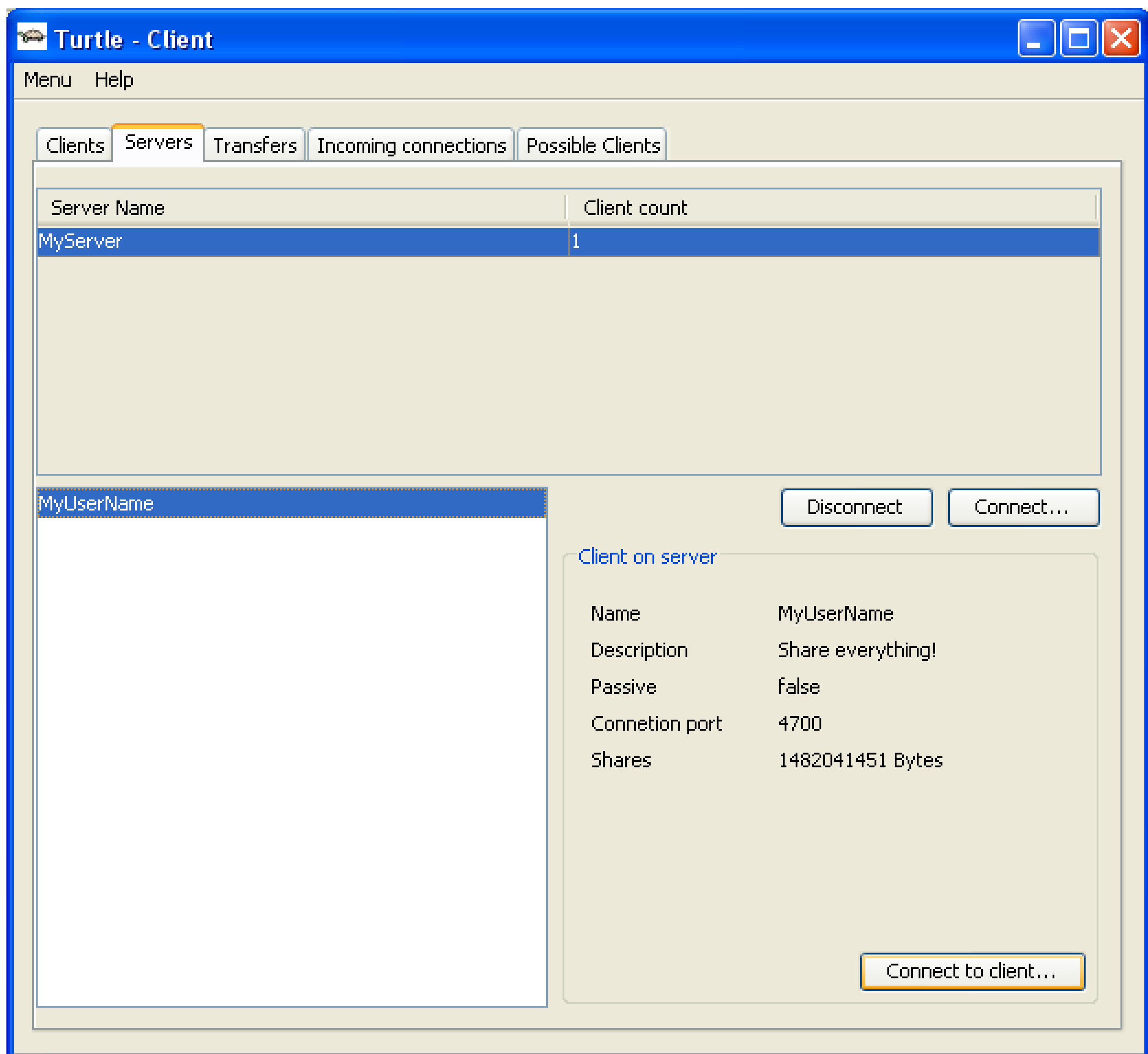
- pripojenie aktívneho klienta na pasívneho – keďže pripojenie rovno na pasívneho klienta nie je možné, v tomto prípade server žiada pasívneho klienta aby sa pripojil na aktívneho. Po úspešnom pripojení ďalšia komunikácia (aj stiahnutia) už prebiehajú bez sprostredkovania serveru.
- komunikácia medzi dvoma pasívnymi klientmi – v tomto prípade všetka komunikácia prebieha prostredníctvom servera.

Pripojenie na nejaký server prebieha pomocou Connection Manager-a, pri pripojení sa na server potrebujeme len adresu servera a port, cez ktorý prebieha komunikácia.



Obr. 18: Pripojenie sa na server

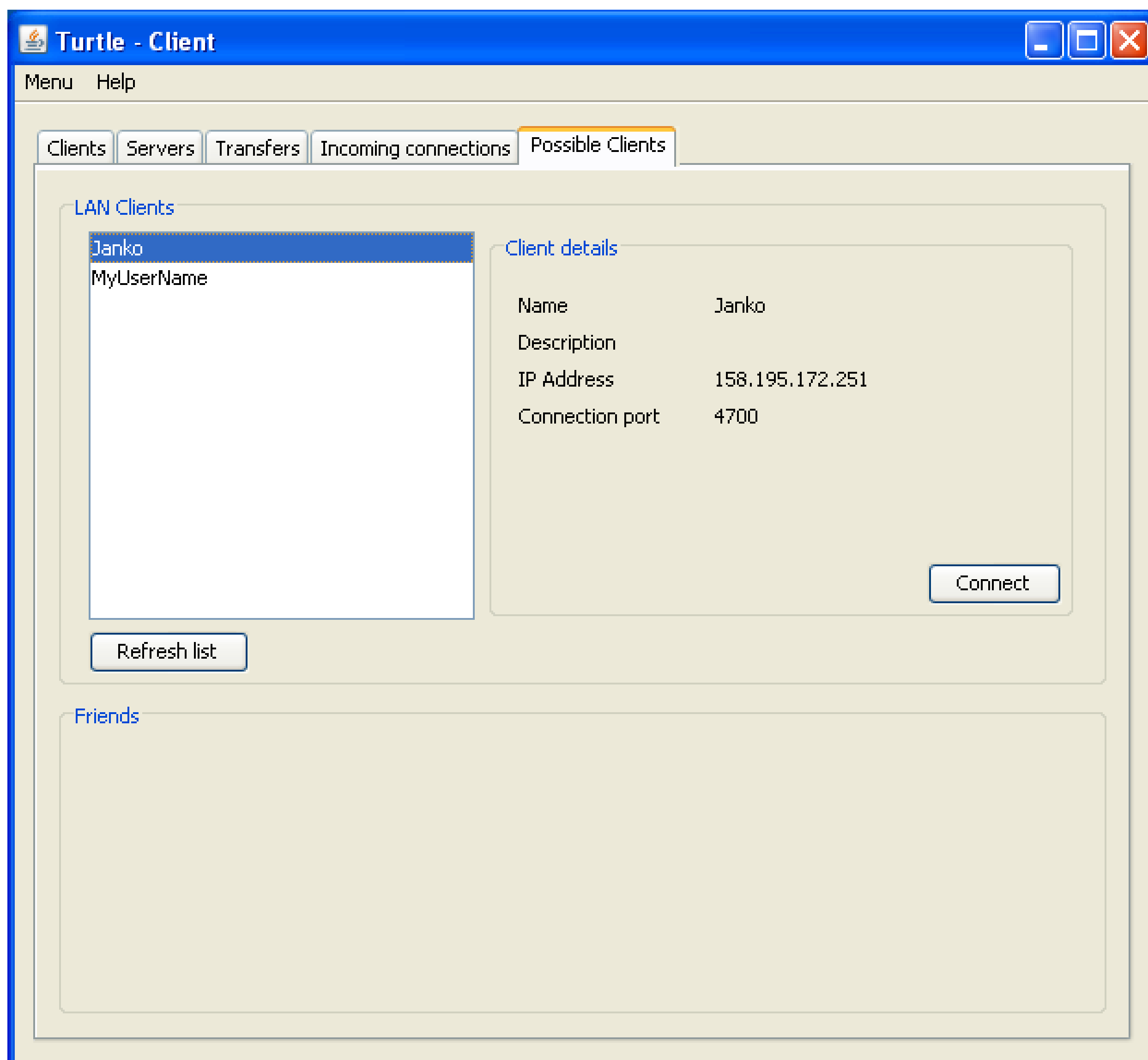
Po úspešnom pripojení na záložke „Servers“ nájdeme v zozname serverov ten, na ktorý sme sa pripojili. Keď vyberieme nejaký server zo zoznamu, objaví sa zoznam klientov pripojených na server a informácie o nich.



Obr. 19: Pripojenie na klienta na serveri.

Keď sa chceme pripojiť na klienta, stlačíme tlačítko „Connect to client...“, a postupujeme identicky ako v prípade pripojenia sa na klienta bez servera.

9. Užívatelia na lokálnej sieti



Obr. 20: Klienti na lokálnej sieti

Klientska aplikácia ponúka možnosť vyhľadávania klientov na lokálnej sieti. Na záložke „Possible clients“ po stlačení tlačítka „Refresh list“ aplikácia nájde všetkých klientov na lokálnej sieti a zobrazí ich v zozname. Ďalej sa na nich už pripájame ľahko, pomocou Connection Managera, podobne ako v prípade klienta s verejnou IP adresou.