MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

Sistem de transfer al fișierelor folosind torrente

LUCRARE DE LICENȚĂ

Absolvent: **Daniel PORAV**

Coordonator: **Șef lucr. ing. Cosmina IVAN**

2011

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

VIZAT,

DECAN, ŞEF CATEDRĂ,

Prof. dr. ing. Sergiu NEDEVSCHI **Prof. dr. ing. Rodica POTOLEA**

Absolvent: **Daniel PORAV**

Sistem de transfer al fișierelor folosind torrente

1. **Enunţul temei:** *Proiectul își propune realizarea unui sistem de transfer al fișierelor în cadrul unei rețele folosind torrente.*

2. **Conţinutul lucrării:** *Cuprins, Introducere, Obiective și specificația proiectului, Studiu bibliografic, Analiză și proiectare de detaliu, Implementare, Testare și validare, Manual de utilizare, Concluzii, Bibliografie, Glosar de termeni, Lista figurilor.*

3. **Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Catedra Calculatoare

4. **Consultanţi:** Șef lucr. ing. Cosmina IVAN

5. **Data emiterii temei:** 1 noiembrie 2010

6. **Data predării:** 24 iunie 2011

Absolvent: _____

Coordonator științific: _____

Declaraţie

Subsemnatul, *Daniel PORAV*, student al Facultăţii de Automatică şi Calculatoare, Universitatea Tehnică din Cluj-Napoca, declar că ideile, analiza, proiectarea, implementarea, rezultatele şi concluziile cuprinse în această lucrare de licenţă constituie efortul meu propriu, mai puţin acele elemente ce nu îmi aparţin, pe care le indic şi recunosc ca atare.

Declar de asemenea că, după ştiinţa mea, lucrarea în această formă este originală şi nu a mai fost niciodată prezentată sau depusă în alte locuri sau alte instituţii decât cele indicate în mod expres de mine.

Data: 24 iunie 2011

Absolvent:**Daniel Porav**

Număr matricol:

Semnătura:_____

Cuprins

1. INTRODUCERE

1.1. Contextul temei

Fiecare din ultimele trei secole a fost marcat de câte o tehnologie. Secolul al XVIII-lea a reprezentat epoca marilor sisteme mecanice însoțite de Revoluția Industrială. Secolul al XIX-lea a fost marcat de motorul cu aburi iar secolul XX a reprezentat un nou pas în evoluția tehnologică prin colectarea, prelucrarea și distribuția informațiilor. Printre multe alte evoluții care au avut loc, am putut observa instalarea rețelelor de telefonie la nivel mondial, inventarea radioului și al televizorului, nașterea și creșterea fără precedent a industriei de calculatoare, precum și lansarea de sateliți de comunicare.

Ca urmare a progresului tehnologic rapid, aceste domenii sunt rapid convergente iar diferențele dintre colectarea, transportul, stocarea și procesarea informațiilor sunt pe cale de dispariție. Companii cu sute de birouri răspândite pe o arie geografică mare se așteaptă să poată examina starea curentă chiar și a celor mai îndepărtate avanposturi la o simplă apăsare a unui buton. Cu cât abilitatea noastră de a aduna, procesa și distribui informații crește, cererea pentru noi tehnici sofisticate de prelucrare a informațiilor crește și mai repede.

Deși industria de calculatoare este încă tânără în comparație cu alte industrii, calculatoarele au înregistrat progrese spectaculoase într-un timp scurt. În primele două decenii ale existenței lor, sistemele informatice au fost foarte centralizate. O companie de dimensiune medie sau o universitate ar fi putut avea unul sau două calculatoare în timp ce o companie mare avea în cel mai bun caz câteva zeci. Ideea că în 20 de ani, calculatoare la fel de puternice și de dimensiunea unui timbru ar fi produse în masă, era ceva pur științifico-fantastic.

Fuziunea calculatoarelor și a sistemelor de comunicații a avut o influență profundă asupra modului în care sunt organizate sistemele informatice. Conceptul de "centru de calcul" ca și o cameră cu un calculator mare la care utilizatorii își aduc propria muncă pentru prelucrare este acum total depășit. Modelul vechi în care un singur calculator servea toate nevoile organizației a fost înlocuit cu un model în care un număr mare de calculatoare separate, dar interconectate, execută taskurile necesare. Aceste sisteme poartă denumirea de rețele de calculatoare.

În continuare termenul de rețea de calculatoare se va referi la o colecție de calculatore autonome și interconectate printr-o tehnologie unică. Conform [1] două calculatoare sunt interconectate dacă sunt capabile să facă schimb de informații între ele. Conexiunea fizică nu trebuie să fie neapărat realizată printr-o sârmă de cupru. Fibra optică, microundele, razele infraroșu și sateliții de comunicare pot fi de asemenea utilizați. Rețelele vin în multe forme și mărimi. Deși sună ciudat pentru unii oameni, nici Internetul și nici World Wide Web-ul (WWW) nu sunt rețele de calculatoare deoarece Internetul nu este o rețea unică, ci mai degrabă o rețea de rețele iar Web-ul este un sistem distribuit care rulează în partea superioară a Internetului.

Există o confuzie considerabilă în literatura de specialitate între o rețea de calculatoare și un sistem distribuit. Distincția principală este reprezentată prin faptul că, într-un sistem distribuit, o colecție de calculatoare independente pare a fiind ,utilizatoriilor săi, un sistem unic și coerent. De obicei, acesta are un singur model sau paradigmă, model care poate fi observat de utilizatori.

Într-o rețea de calculatoare, această coerență, model și software sunt absente. Utilizatorii sunt expuși la calculatoare reale, fără nici o încercare a sistemului de a face ca mașinile să arate sau să funcționeze într-un mod coerent.

Într-adevăr, un sistem distribuit este un sistem software construit pe nivelul superior a unei rețele. Software-ul oferă un grad înalt de coerență și transparență. Astfel, distincția dintre o rețea și un sistem distribuit este reprezentată mai degrabă de software-ul, în special de sistemul de operare, decât de parte de hardware.

Cu toate acestea, există o suprapunere considerabilă între cele două discipline. De exemplu, ambele sisteme, cele distribuite și rețelele de calculatoare au nevoie să transfere diferite fișiere. Diferența constă în componenta care invocă transferul, sistemul sau utilizatorul.

1.2. Conturarea domeniului

- Începutul BitTorrentului

Protocolul BitTorrent folosit în rețelele peer-to-peer (P2P) a fost elaborat aproximativ în același timp când modelul anterior de partajare a fișierelor în rețelele P2P a fost aproape de sfârșit. Declinul clienților P2P, cum ar fi Napster și eDonkey a fost marcat de reinventarea Napster-ului, cea mai mare rețea P2P la acel moment, ca un magazin de muzică online, ca urmare a unei serii de procese legale împotriva ei și a altor rețele P2P de către deținătorii drepturilor de autor. În momentul în care titularii drepturilor de autor au acționat în justiție autorii aplicației, popularitatea primei generații de rețele P2P a început să decadă. A doua generație era deja în așteptare și gata să preia ștafeta.

"Mișcarea" BitTorrent, odată stabilită în domeniul Internetului, a reușit să reunească o mulțime de utilizatori aproape instantaneu. Creșterea rapidă în popularitate a rețelei a venit ca urmare a renunțării utilizatoriilor la vechile rețele, a închiderii unor rețele sau ca urmare a calității slabe a clienților P2P existenți. Furnizorii BitTorrent nu au avut doar noroc, aceștia au presimțit ceea ce urma să vină și s-au pregătit pentru reacția împotriva primei generații de rețele P2P iar apoi au procedat prin punerea în aplicare a unui serviciu care a fost de departe superior față de ceea ce s-a putut vedea anterior.

Ei au încercat să înțeleagă ce a mers prost în cazul primei generații de programe de tipul file-sharing și au evitat să facă aceleași greșeli din nou. Furnizorii de clienți și-au minimizat expunerea lor la problemele de drepturi de autor, au adus noi instrumente lipsite de spy-ware și ad-ware și au făcut sistemul astfel încât să nu fie posibilă utilizarea acestuia în rețelele de dimensiuni mari și independente.

- Legalitatea sistemului

Prima generație de programe adresate rețelelor P2P au venit cu funcții de căutare incluse pentru a putea afla ce fișiere partajează fiecare utilizator în parte. Informațiile returnate erau indexate și stocate pe serverele diferitelor rețele (eDonkey), pentru a accelera procesul de căutare. Prin stocarea informațiilor respective a fost posibilă aducerea proceselor legale împotriva rețelelor P2P, deoarece stocau informații care permiteau descărcarea de material protejat de drepturi de autor. Noua generație de clienți P2P nu stochează asemenea informații, în schimb se bazează în întregime pe capacitatea utilizatorilor să obțină informațiile respective de la numeroasele site-uri independente iar apoi de la serverele de tracking independente. În acest sens, este mult mai greu să se aducă învinuiri împotriva celor care oferă clienții de torrente, iar trackerele si site-urile din acest domeniu pot fi mutate cu ușurință cu scopul de a evita legile unei anumite țări, ceea ce Pirate Bay deja a dovedit. Desigur, acest lucru a dus la o schimbare semnificantă pentru deținătorii drepturilor de autor. În loc să se axeze pe furnizorii de clienți de torrente, posesorii drepturilor de autor își îndreaptă atenția spre utilizatorii care descarcă concret datele.

- Următoarea generație de clienți de torrente

Azureus Inc., producătorii unuia dintre cei mai populari clienți de torrente, Vuze (inițial denumit Azureus), a făcut deja primi pași în crearea următoarei generații de clienți de torrente, respectiv aplicații de file-sharing. Acești pași se bazează pe descărcări comeriale plătite pentru a genera un flux de venituri și a putea plasa produsele într-un conținut gratuit. De asemenea, se pare că, conținutul de circulație oferit comercial de Azureus este reprezentat în primă fază de videoclipurile lungi și de calitate foarte bună, asemănătoare celor de pe YouTube. Dacă viteza de partajare a fișierelor va deveni în viitor mult mai rapidă, clienții de torrente ar putea devenii principalii furnizori de video pe Internet, deoarece va fi în măsură să ofere "produse" de calitate foarte bună la costuri mici în ceea ce privește resursele unei rețele cum ar fi lățimea de bandă.

2. OBIECTIVELE ȘI SPECIFICAȚIA PROIECTULUI

2.1. Tema propriu-zisă

Având în vedere existența unei multitudini de sisteme asemănătoare și complexitatea acestora, s-a hotărât dezvoltarea unui sistem simplu de transfer a informațiilor digitale într-o rețea de calculatoare cu ajutorul torrentelor. Sistemul cuprinde două aplicații, una care reprezintă actorul principal al sistemului, și anume peer-ul, iar cealaltă care face posibilă comunicarea între peer-urile care se află în aceeași rețea, și anume trackerul.

Obiectivul principal al acestui proiect este reprezentat de încercarea de parcurgere a tuturor pașilor de creare a unui produs software în cel mai profesional mod posibil. De asemenea, s-a încercat includerea, în acest proces, a cât mai multe cunoștințe dobândite pe parcursul facultății demonstrându-se astfel capacitatea de proiectare, implementare și modificare acolo unde este cazul, a unui produs software.

Proiectul de față reprezintă un sistem de transfer al fișierelor în cadrul unei rețele folosind fișiere speciale de metadată, denumite fișiere torrent. Proiectul constă în două aplicații: o aplicație cu rol de client, folosită de utilizatorul obișnuit, prin care acesta poate să descarce date de la alți utilizatori ai aplicației din aceeași rețea si o aplicație cu rol de server, numită tracker, care face rutarea datelor in cadrul rețelei respective.

De asemenea, s-a încercat includerea a cât mai multor tehnologii și componente existente în crearea proiectului, ușurându-se astfel munca și utilizând diferitele componente cu scopul pentru care au fost create. Utilizarea diferitelor librării, de exemplu, care au fost folosite în proiect a presupus în primul rând găsirea acestora, determinarea modului lor de funcționare și integrarea lor în cadrul proiectului. Această etapă este considerată a fii una din cele mai importante etape în procesul de creare software deoarece, practic, acesta este primul pas principal și realizând această etapă într-un mod corect, riscul proiectului scade semnificativ.

2.2. Cerințele funcționale

Conform [2] cerințele funcționale ale unui sistem determină funcțiile acestuia sau a unei componente ale acestuia. Prin funcție înțelegem mulțimea datelor de intrare, comportamentul sistemului și mulțimea datelor de ieșire.

Principalele cerințe de funcționalitate ale aplicației client din cadrul proiectului de față sunt:

- Crearea fișierelor de tip torrent într-un mod cât mai simplu;
- Publicarea torrentelor pe orice tracker activ;
- Descărcarea datelor folosind un torrent creat correct;
- Determinarea diferițiilor parametri de setare ale aplicației;
- Gestionarea eficientă a torrentelor active.

Cerințele funcționale ale componentei de tracking sunt:

- Vizualizarea traficului și a diferitelor informații despre utilizatori și torrentele publicate pe trackeul respectiv în cadrul rețelei în care operează acesta;

- Crearea diferitelor rapoarte pe baza acestor informații;
- Determinarea diferiților parametri de setare ale aplicației
- Posibilitatea de interzicere a utilizării trackerului de anumiți utilizatori.

2.3. Cerințele non-funcționale

Cerințele non-funcționale ale unui sistem software specifică criteriile care se pot folosi pentru a implementa cerințele funcționale. Cerințele non-funcționale pentru aplicația client sunt:

- Utilizabilitatea: aplicația client trebuie să fie ușor de învățat prin intermediul unei interfețe simple care să permită utilizatorilor săi să folosească cât mai rapid și într-un mod cât mai eficient aplicația. De asemenea, aplicația trebuie să ofere un grad ridicat de satisfacție din acest punct de vedere;
- Portabilitatea: aplicația client trebuie să fie independentă atât față de platforma sistemului de operare pe care rulează cât și față de modelul procesorului existent pe acesta;
- Extensibilitate: aplicația client trebuie să fie extensibilă pentru a putea adăuga în viitor diferite funcționalități suplimentare sau pentru a îmbunătăți calitatea celor existente.

Cerințele non-funcționale pentru aplicația de tracking sunt:

- Accesul concurent: aplicația de tracking trebuie să permită un acces concurent cât mai multor utilizatori deoarece numărul de utilizatori este direct proporțional cu disponibilitatea fișierelor transferate și cu lățimea de bandă existentă;
- Securitate: cel puțin din punctul de vedere al aplicației de tracking, sistemul trebuie să aibă un grad minimal de securitate. Datele transmise între clienții sistemului și trackerul care gestionează traficul fișierelor în rețeaua respectivă trebuie să fie criptate;
- Scalabilitate: practic, scalabilitatea reprezintă o non-funcționalitate a întregului sistem, nu numai a aplicației de tracking, însă datorită faptului că aplicația reprezintă partea centralizată a sistemului am decis să fie specificată în acest loc. Sistemul trebuie să fie scalabil, deoarece, cum s-a specificat mai devreme, un număr mai mare de utilizatori activi reprezintă o performanță mai ridicată a întregului sistem.

2.4. Cazuri de utilizare

Conform [12], în ingineria software și ingineria sistemelor, un caz de utilizare reprezintă o descriere a pașilor sau a acțiunilor care au loc între un utilizator (sau „actor”) și un sistem software care conduce utilizatorul spre un obiectiv stabilit. Utilizatorul poate fi reprezentat de o persoană sau despre un obiect abstract, cum ar fi un sistem software extern sau un proces manual. În termen de inginerie de sistem, cazurile de utilizare sunt folosite la un nivel superior decât în termen de inginerie software, reprezentând adeseori taskuri sau obiective a părților interesate.

Un caz de utilizare este o tehnică de modelare folosită pentru a descrie ce va face un sistem nou sau ce face deja un sistem existent. Scopul urmărit în acest tip de modelare este reprezentat de descrierea funcționalității sistemului așa cum acesta este văzută din exterior de

un număr de actori și a conexiunilor acestora la cazurile de utilizare furnizate de sistem. Așadar, pentru crearea modelului cazurilor de utilizare vor trebui identificați actorii, cazurile de utilizare, relațiile dintre acestea cât și relațiile acestora cu actorii.

2.4.1. Scriere cazurilor de utilizare

După cum se poate vedea și în numele sub-capitolului, [13] cazurile de utilizare se scriu și nu se desenează deoarece, acestea din urmă nu sunt diagrame. Cazurile de utilizare sunt mai degrabă descrieri textuale ale cerințelor comportamentale scrise dintr-un anumit punct de vedere.

Un caz de utilizare mai poate fi văzut și ca o descriere comportamentală a unui sistem. Descrierea este scrisă dintr-un punct de vedere al utilizatorului care tocmai a spus sistemului să facă un anumit lucru. Un caz de utilizare capturează secvența vizibilă a evenimentelor prin care trece un sistem ca răspuns la un stimul singular din partea utilizatorului.

Un eveniment vizibil, reprezintă un eveniment vizibil de către utilizator. Cazurile de utilizare nu descriu comportamente sau mecanisme ascunse, din punctul de vedere al utilizatorului, în nici o măsură. Descriu doar lucrurile pe care un utilizator le poate vedea.

2.4.2. Diagramele cazurilor de utilizare

Scopurile principale al construirii acestui tip de diagramă sunt:

- Să decidă și să descrie cerinele funcționale ale sistemului, cerințe care au fost deduse după o discuție între client și/sau utilizatorii sistemului și viitorii dezvoltatori ai acestuia;
- Să ofere o descriere clară și consistentă a ce va trebui să facă sistemul, prin urmare modelul este folosit pentru comunicarea cerințelor tuturor persoanelor implicate în construirea sistemului și să constituie un punct de plecare în realizarea muncii viitoare (alte metode, design arhitectural, implementarea propriu-zisă);
- Să constituie o bază pentru realizarea testelor de verificare dacă funcționalitatea finală a sistemului concordă cu cerințele inițiale ale acestuia;
- Să permită o transformare ușoară a cerințelor funcționale în viitoare clase si operații;

Conform [14], o diagramă a unui caz de utilizare reprezintă o imagine a contextului unui sistem. O diagramă bună a unui caz de utilizare arată printre altele limitele sistemului descris, ce se află în afara acestor limite și cum se folosește sistemul respectiv. Practic, diagramele cazurilor de utilizare sunt instrumente de comunicare ce rezumă comportamentul unui sistem și a actorilor acestuia.

În acestă fază de modelare sistemul este văzut ca o "cutie neagră", care trebuie să poată realiza anumite sarcini, fără a ne interesa cum le face, cum vor fi implementate, sau cum lucrează intern.

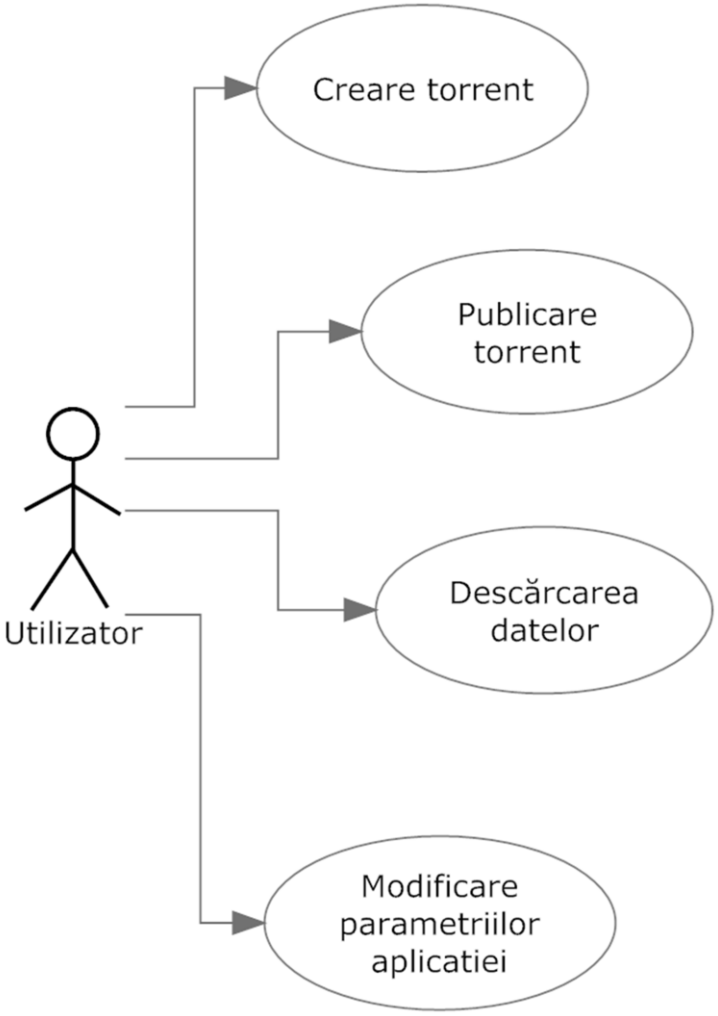

Figura 2.1. Diagrama cazurilor de utilizare a aplicației client

În continuare vor fi descrise pe scurt cazurile de utilizare:

- Specificație de caz de utilizare: *Creare torrent*

Acest caz de utilizare începe în momentul în care utilizatorul deschide aplicația și selectează opțiunea de creare a unui torrent.

1. Utilizatorul introduce datele necesare creării torrentului: nume, cale, dimensiunea pachetelor torrentului respectiv, fișierele conținute de acesta, URL-ul trackerului pe care va fi publicat, creatorul și un comentariu opțional;
2. Utilizatorul apasă butonul de creare a torrentului;
3. Utilizatorul primește un mesaj de tipul pop-up prin care este anunțat dacă torrentul s-a creat cu succes sau a avut loc o eroare în procesul de creare a acestuia.

- Specificație de caz de utilizare: *Publicare torrent*

1. Utilizatorul selectează opțiunea de publicare a unui anumit torrent;

2. Utilizatorul introduce datele necesare publicării torrentului și selectează torrentul respectiv: URL-ul trackerului pe care va fi publicat torrentul, numele creatorului și un comentariu opțional;
3. Utilizatorul apasă butonul de publicare a torrentului;
4. Utilizatorul primește un mesaj de tipul pop-up prin care este anunțat dacă torrentul respectiv a fost publicat cu succes sau nu.

- Specificație de caz de utilizare: *Descărcarea datelor*

1. Utilizatorul selectează opțiunea de deschidere a unui torrent;
2. Utilizatorul selectează torrentul care conține informațiile despre fișierele dorite de utilizator;
3. Aplicația începe să descarce fișierele dorite, în cazul în care există donatori sau parteneri disponibili.

- Specificație de caz de utilizare: *Modificarea parametriilor aplicației*

1. Utilizatorul deschide fereastra de setări prin apăsarea butonului corespunzător acestei acțiuni;
2. Utilizatorul modifică parametrii doriți;
3. Utilizatorul salvează modificările efectuate și închide fereastra de setări.

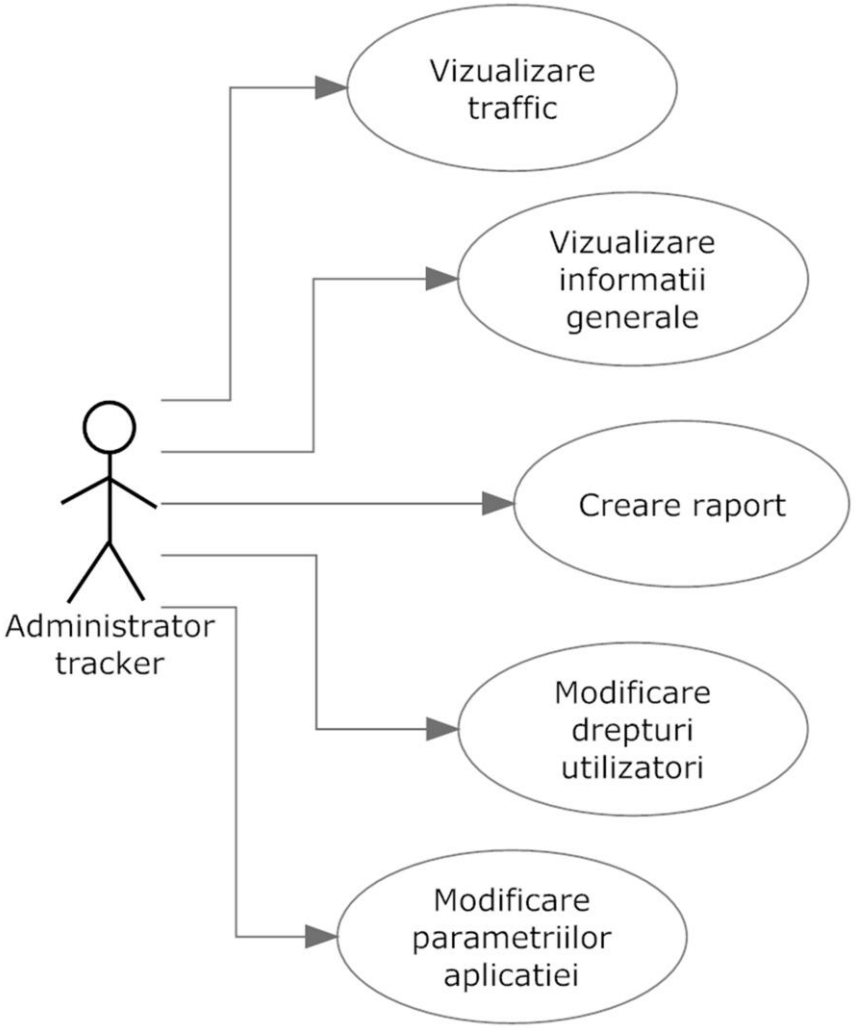

Figura 2.2. Diagrama cazurilor de utilizare a aplicației tracker

În continuare vor fi descrise pe scurt cazurile de utilizare ale aplicației tracker:

- Specificație de caz de utilizare: *Vizualizare trafic*

1. Administratorul de tracker selectează opțiunea de vizualizare a traficului în timp real din cadrul rețelei din care face parte;
2. Datele sunt afișate în timp real într-o fereastră nouă.

- Specificație de caz de utilizare: *Vizualizare informații generale*

1. Administratorul de tracker selectează opțiunea de vizualizare a informațiilor generale a unui torrent sau peer, din cadrul rețelei;
2. Administratorul de tracker introduce datele necesare selectării torrentului sau perr-ului respectiv și apasă butonul de afișare.

- Specificație de caz de utilizare: *Creare raport*

1. Administratorul de tracker selectează opțiunea de creare a unui raport;
2. Administratorul de tracker introduce datele necesare creării raportului respectiv și apasă butonul de creare a raportului.

- Specificație de caz de utilizare: *Modificare drepturi utilizatori*

1. Administratorul de tracker selectează peer-ul dorit;
2. Administratorul de tracker modifică drepturile peer-ului în cadrul sistemului.

- Specificație de caz de utilizare: *Modificare parametriilor aplicației*

1. Administratorul de tracker deschide fereastra de setări prin apăsarea butonului adiacent acestei opțiuni;
2. Administratorul de tracker modifică parametrii doriți;
3. Administratorul de tracker salvează modificările efectuate și închide fereastra de setări.

3. STUDIU BIBLIOGRAFIC

3.1. Rețelele peer-to-peer

Rețelele și tehnologia peer-to-peer (P2P) reprezintă una din cele mai importante evoluții pentru arhitectura sistemelor distribuite mari și a Internetului. Aplicațiile utilizate pe scară largă au dovedit potențialul fezabil și economic al tehnologiei pentru servicii care implică milioane de utilizatori. Tehnologia P2P este o paradigmă în curs de dezvoltare care, în acest moment, este văzută ca un posibil înlocuitor pentru re-proiectarea arhitecturiilor distribuite.

Într-o rețea P2P clasică, toate calculatoarele, numite noduri, au atât capacități cât și responsabilități echivalente. Nodurile pot schimba între ele resurse și apela servicii fără a fi necesară existența unor servere centralizate. De asemenea, nodurile pot colabora pentru a îndeplini diferite taskuri prin utilizarea în comun a resurselor (spațiului de stocare, puterii de calcul a UPC-urilor, etc.) din cadrul rețelei P2P.

Modelul P2P se diferențiază de clasicul model distribuit client-server în 3 mari aspecte:

- În primul rând, scalabilitatea rețelelor P2P poate ajunge mult mai departe față de sistemele distribuite clasice. Astfel, prin capacitatea sistemelor P2P de a ”crește” la mii de noduri, acestea pot valorifica puterea calculatoarelor pe Internet;
- În al 2-lea rând, tehnologia P2P, în cea mai intransigentă definiție, cere ca totul să fie complet decentralizat. În mod ideal, într-un sistem P2P nu ar trebui să existe nici o structură centralizată;
- În utimul rând, cea mai importantă diferență este reprezentată de modul de lucru al sistemelor P2P în medii dinamice. Concret, în ceea ce privește topologia rețelei, deoarece nodurile se pot alătura și părăsi rețeaua în orice moment, sistemele P2P nu au o topologie fixă. În schimb, topologia acestora se schimbă în funcție de nodurile din rețea.

Dimensiunea și dinamismul care caracterizează sistemele P2P necesită o reexaminare a tehnologiilor distribuite clasice. Este nevoie de o trecere la o paradigmă care include auto-organizare, adaptare și “elasticitate”. În ultimii ani, a existat o proliferare a eforturilor de cercetare în proiectarea sistemelor și aplicațiilor P2P.

3.2.1. Tipuri de aplicații P2P

Aplicațiile P2P pot fi împărțite în 2 mari categorii: aplicații cu partajare a resurselor și aplicații cu partajare a datelor. În cazul partajării resurselor, aplicațiile permit nodurilor să folosească resurse, precum, procesoare, stocare pe disc sau lățime de bandă, resurse disponibile în rețeaua P2P din care fac parte. Modelul P2P permite valorificarea resurselor neutilizate din cadrul unei rețele pentru a efectua sarcini care altfel, ar necesita o mașină mult prea scumpă, cum ar fi un super calculator.

În cazul partajării datelor, aplicațiile permit utilizatoriilor să acceseze, să modifice și să interschimbe date într-o manieră flexibilă. Față de partajarea resurselor de stocare a datelor, partajarea datelor implică atât stocare cât și privilegii de accesare, sisteme de notificare automate și tehnici de multicast și broadcast.

3.2. Protocolul BitTorrent

[7] BitTorrent este un protocol P2P de partajare a fișierelor utilizat pentru a distribui cantități mari de date în cadrul unei rețele de calculatoare. Protocolul poate fi folosit pentru a reduce impactul pe care îl are distribuirea datelor asupra serverului și a rețelei în sine. În loc să descarce un fișier de la o singură sursă, și anume de la server, protocolul BitTorrent permite utilizatoriilor să se alăture unei mulțimi de hosturi pentru a încărca și descărca simultan date între ei. Protocolul reprezintă o alternativă la tehnica clasică de distribuire a datelor și poate funcționa și în rețele cu lățime de bandă mică, astfel chiar și mini calculatoarele, cum ar fi telefoanele mobile sau tablet-urile, sunt capabile să distribuie date la mai mulți destinatari.

3.2.1. Termeni generali

Principalii termeni generali ai protocolului sunt:

- Torrentul - Fișierul .torrent nu este fișierul care trebuie transferat. Acesta este un fișier foarte mic care conține informații despre fișierul respectiv și despre utilizatorii care partajează datele respective. Poate fi comparat cu o hartă folosită de clientul de torrente pentru a asambla toate fragmentele fișierului transferat;
- Clientul de torrente – Clientul de torrente reprezintă una din cele mai importante părți al sistemul de transfer al fișierelor folosind torrente. Această componentă ia fișierul torrent, citește informațiile din fișier și descarcă datele necesare;
- Peer – Peerul reprezintă un nod al rețelei de calculatoare care participă în procesele de descărcare și încărcare a unui fișier .torrent;
- Donator (eng. Seeder) – Un donator este un peer al rețelei care deține o copie completă a fișierului partajat în rețea;
- Partener (eng. Leecher) – Un partener este un peer al rețelei care nu deține, încă, întregul fișier partajat. Partenerul devine donator în momentul în care a terminat de descărcat întregul fișier partajat și începe să îl ofere și celorlalți utilizatori;
- Raportul de partajare (eng. Share ratio) – reprezintă raportul dintre cantitatea de date pe care un utilizator a încărcat-o și cantitatea de date pe care a descărcat-o pentru un torrent particular. Un raport de partajare mai mare ca 1 are un efect pozitiv asupra "reputației" utilizatorului deoarece acest lucru indică faptul că, cantitatea de date încărcate este mai mare decât cantitatea de date descărcate. În mod similar, un raport de partajare mai mic decât 1 are un efect negativ;
- Multime (eng. Swarm) – reprezintă numărul total de donatori și parteneri care participă în procesul de transfer al fișierelor folosind torrente;
- Tracker – Trackerul este o componentă cu rol de server, care deține informații despre utilizatorii sistemului și despre fișierele transferate între aceștia. Practic, trackerul acționează ca un bridge între parteneri și donatori. Trackerul poate fi privat, utilizatorul care dorește să îl folosească având nevoie să se înregistreze în prealabil.

3.2.2. Modul de funcționare

Orice client de torrente este capabil să pregătească, să solicite și să transmită orice fel de fișier în cadrul unei rețele folosind protocolul BitTorrent.

Pentru a putea transmite un fișier sau o multime de fișiere, utilizatorul trebuie, în primă fază, să creeze fișierul torrent. Fișierul conține metadate despre fișierele partajate și

despre trackerul folosit pentru a transfera datele respective. Utilizatorii care doresc să descarce datele partajate trebuie să obțină mai întâi fișierul torrent și să se conecteze la trackerul specificat în acesta pentru a putea găsi utilizatori care partajează datele necesare.

Față de protocoalele clasice, cum ar fi HTTP (Hypertext Transfer Protocol) sau FTP (File Transfer Protocol), protocolul BitTorrent se diferențiază prin 2 mari aspecte:

- Protocolul BitTorrent împarte datele transferate în fragmente, transferând pe urmă aceste fragmente folosind mai multe conexiuni TCP (Transmission Control Protocol) între diferite peeruri, în timp ce transferul clasic se face folosind doar o singură conexiune;
- Și protocolul BitTorrent descarcă fragmentele într-o ordine aleatoare, lucru ce determină o mai mare disponibilitate a datelor transferate, în timp ce transferul clasic este secvențial.

3.3. API-ul Java Bittorrent

[3] Java Bittorrent API este un API (Application Programming Interface) simplu și ușor de folosit, care permite oricărui dezvoltator software să creeze aplicații Java care necesită utilizarea protocolului Bittorrent, descris mai devreme. În crearea API-ului s-au folosit diferite clase și librării existente pentru a facilita dezvoltarea software-ului și datorită eficienței și ușurinței în utilizare a acestora.

- Librăria "Simple Web"

În cazul în care un utilizator dorește să pună un tracker online, va avea nevoie de un server web care să găzduiască fișierele necesare și să răspundă cererilor clienților. Librăria "Simple Web" asigură posibilitatea executării codului Java, și anume a serviciilor disponibile, cu scopul de a răspunde cererilor clienților. Pe de altă parte , librăria este mai ușor de folosit, comparativ cu alte soluții, cum ar fi Tomcat, atâta timp cât asigură o performanță la fel de bună.

- Librăria "JDOM"

Trackerul integrat în API nu folosește o bază de date pentru a memora informațiile despre torrente și clienți. În schimb, se bazează pe fișiere XML. Pentru a gestiona fișierele XML, API-ul folosește librăria "JDOM" care asigură o soluție completă, bazată pe Java pentru a accesa, manipula și scrie date în fișiere XML folosind cod Java.

- Clasa "ClientHTTPRequest"

Comunicarea dintre clienți și trackere pentru încărcarea fișierelor folosește un tip special de cerere HTTP: *multipart/form-data*. Aceste cereri sunt codificate într-un mod diferit pentru a permite transferul de fișiere odată cu textul sau cu alți parametrii transmiși.

Clasa ClientHTTPRequest implementează acest protocol și conține diferite metode care permit, într-un mod simplu și eficient, adăugarea de conținut la o cerere și trimiterea ei la un server web.

- Clasele "BEncoder" și "BDecoder"

Protocolul BitTorrent folosește o codificare specială pentru torrente și pentru răspunsurile trackerului, numită codificare *BEncoding*. Există multe clase Java care implementează acest tip de codificare, printre ele aflându-se și clasele "BEncoder" și "BDecoder".

3.4. Tehnologiile folosite

3.4.1. Java

[4] Java este un limbaj de programare orientat obiect consacrat. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile gen telefon, agendă electronică, tabletă, etc. În felul acesta se creează o platforma unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Avantajele sunt evidente, atât pentru proiectanți, care "scriu odată și execută pe orice mașină virtuală Java (JVM)", cât și pentru utilizatori, care vor beneficia de un spectru îmbogățit de servicii. De aceea se poate afirma cu încredere că Java este un câștigător, în lumea volatilă a tehnologiilor de calcul.

Java este un limbaj de programare de nivel înalt, orientat obiect, proiectat inițial pentru realizarea de aplicații pentru Internet și mai cu seamă pentru Web. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor. În acest sens, multe companii recurg la limbajul Java în procesul de informatizare întrucât oferă un foarte bun suport pentru tehnologiile de vârf și, nu în ultimul rând, pentru faptul că este gratuit și în mod continuu îmbogățit și îmbunătățit.

Conform [4] caracteristicile limbajului Java sunt:

- *Limbaj interpretat și compilat*. Un limbaj este înterpretat dacă instrucțiunile unui program scris în acel limbaj sunt procesate linie cu linie și traduse în cod mașină. Un limbaj este compilat dacă un program scris în acel limbaj este tradus într-un cod pe care calculatorul îl poate "înțelege" și executa mult mai ușor. Programele interpretate sunt mai lente decat cele compilate, insa cele compilate sunt de obicei dependente de platforma respectiva. Programele Java sunt mai întâi compilate în niște fișiere intermediare asemănătoare codului de asamblare, numite "byte code", apoi acestea sunt interpretate de mediul de execuție Java în instrucțiuni mașină asociate platformei sistem;

- *Limbaj independent de platformă*. La instalarea limbajului Java, se va crea o mașină virtuală Java care are drept scop traducerea instrucțiunilor unui "byte code" Java în instrucțiuni mașină pentru platforma curentă. Astfel fișierele intermediare "byte code" pot fi copiate și executate pe orice platformă, indiferent că este Windows, Unix, Solaric etc.;

- *Limbaj orientat obiect*. Cea mai importantă proprietate a limbajului Java este orientarea obiect, aceasta fiind privită ca o trăsătură implicită. Java pune în evidență toate aspectele legate de programarea orientată obiect: obiecte, trimitere de parametri, încapsularea, clase, biblioteci, moștenire și modificatori de acces;

- *Limbaj concurent*. Concurența (eng. multithreading) reprezintă capacitatea unui program de a executa mai multe secvențe de cod în același timp. O secvență de cod Java se numește fir de execuție (eng. thread). Datorită posibilității creării mai multor fire de execuție, un program Java poate să execute mai multe sarcini simultan, de exemplu animația unei imagini, transmiterea unei melodii spre placa de sunet, comunicarea cu un server, alocarea și eliberarea memoriei, etc.;

- *Limbaj simplu*. Spre deosebire de C++, care este cel mai popular limbaj orientat obiect, Java elimină câteva dintre trăsăturile acestuia: posibiliatatea moștenirii multiple este exclusă, șirurile sunt încapsulate într-o structură clasă. Spre deosebire de C/C++, în Java nu există pointeri, iar alocarea și dealocarea memoriei se face în

mod automat. Există un mecanism de eliberare a memoriei (eng. garbage collection) pus în practică de un fir de execuție de prioritate mică;

- *Limbaj distribuit*. Java este distribuit deoarece permite utilizarea obiectelor locale și de la distanță. Limbajul Java oferă posibilitatea dezvoltării de aplicații pentru Internet, capabile să ruleze pe platforme distribuite și eterogene. În acest sens, Java respectă standardul IEEE (Institute of Electrical and Electronics Engineers) pentru structurile de date, cum ar fi folosirea întregilor, numerelor în virgulă flotantă și șirurilor de caractere. Java se poate utiliza în aplicații de rețea, deoarece respectă protocoalele de rețea, cum ar fi HTTP, FTP, etc.;

- *Limbaj performant*. Interpretorul Java este capabil să execute un byte code aproape la fel de reprede ca un cod compilat. Având posibilitatea să lucreze cu fire de execuție multiple, Java justifică faptul că este un limbaj performant. De exemplu, un program Java poate aștepta citirea unor date, în timp ce un alt fir de execuție poate aloca sau elibera memoria necesară programatorului;

- *Limbaj dinamic și robust*. Java intârzie alocarea obiectelor și legarea dinamică a claselor pentru momentul execuției. Astfel, se vor evita erorile de alocare chiar dacă mediul s-a schimbat de la ultima compilare a programului. Faptul că Java este robust se justifică prin eliminarea utilizării pointerilor, care erau generatoare de erori în cazul programelor C/C++. În schimb, Java verifică memoria dinamic înainte de a fi alocată și are un sistem automat de alocare/dealocare a memoriei;

- *Limbaj sigur*. Programele Java nu pot accesa memoria heap, stack sau alte secțiuni protejate de memorie, deoarece Java nu folosește pointeri și alocă memorie doar la execuție. Înainte ca interpretorul Java să execute "byte code"-ul, se verifică dacă este un cod Java valid prin cercetarea accesului la date, conversiilor de date nepermise, valori și parametri incorecți, depășirea stivei.

3.4.2. ActionScript

[5] ActionScript reprezintă limbajul oficial de programare al platformei Flash de la Adobe. Inițial fiind conceput ca un simplu tool pentru controlul animației, ActionScript a evoluat într-un limbaj de programare sofisticat folosit pentru crearea conținutului și aplicațiilor web, dispozitivelor mobile și crearea aplicațiilor desktop. ActionScript poate fi folosit în diferite modalități, de diferitele tipuri de programatori și producători de aplicații. De exemplu, un animator poate folosi doar câteva linii de cod pentru a opri temporar o animație web. De asemenea, un dezvoltator software poate folosi câteva sute de linii de cod ActionScript pentru a adăuga interactivitate unei interfețe pe un dispozitiv mobil. Un developer poate folosi câteva mii de linii de cod ActionScript pentru a crea o întreaga aplicație de citire a e-mailurilor pentru un browser web și desfășurare (eng. deployment) pe desktop.

ActionScript 3.0 este un limbaj de programare orientat obiect folosit la crearea aplicațiilor și conținutului multimedia care rulează în clienții de rulare Flash (FlashPlayer și Adobe AIR). Având o sintaxă asemănătoare cu cea a limbajelor Java și C#, ActionScript ar trebui să fie familiar programatorilor experimentați. Limbajul ActionScript 3.0 este bazat pe specificațiile limbajului ECMAScript, ediția a 4-a, care este în proces de dezvoltare.

Deși noua versiune al limbajului intern de scripting Flash conține majoritatea caracteristicilor versiunilor anterioare ale limbajului, ActionScript 3.0 trebuie considerat un limbaj de programare nou din câteva motive simple. În primul rând, câteva aspecte diferă

între ultimele versiuni ale limbajului, cum ar fi modelul de evenimente și modul de afișare al elementelor grafice (eng. assets). În al 2-lea rând, mici modificări au fost făcute în interiorul limbajului, modificări care necesită o mare atenție până vor deveni cunoscute dezvoltatorilor. Aceste modificări sunt minore, cum ar fi schimbări ale numelor unor proprietăți. Cel mai important aspect al noii versiuni de ActionScript este acela că a fost rescris în totalitate și foloseste o bază diferită față de celelalte versiuni ale limbajului. Aceste optimizări asigură o creștere mare a performanței aplicațiilor dar împiedică utilizarea ActionScript-ului 3.0 cu alte versiuni anterioare ale limbajului în același fișier.

Principalele diferențe între ultimele versiuni ale limbajului ActionScript sunt:

- *Raport de erori mai detaliat*. ActionScript 3.0 solicită o corectitudine mare a tipurilor de variabile, a argumentelor, a tipului returnat de funcție, etc. Verificarea tipurilor datelor a fost introdusă în ActionScript 2.0 fiind inițial opțională. Verificarea sporită a tipurilor de date îmbunătățește detectarea acestora și oferă mai multe informații cu privire la acestea în timpul scrierii codului pentru a permite o corectare mai rapidă a problemei întâlnite. Această modificare îmbunătățește performanța și reduce memoria necesară deoarece tipurile de date sunt stocate în codul mașină și nu mai necesită adresare dinamică in timpul rulării (eng. runtime);

- *Îmbunătățiri ale sintaxei*. Diferite probleme ale sintaxei versiunilor anterioare au fost rezolvate în ultima versiune. De exemplu, în unele cazuri, numele proprietăților au fost clarificate. De asemenea, diferitele moduri de abordare a operațiilor, cum ar fi încărcarea unei clase externe sau conectarea la un URL s-au constantizat;

- *O nouă arhitectură a display-ului*. Multele metode anterioare de a adăuga elemente în mediul de afișare s-au consolidat. Noua listă de afișare simplifică acest proces și de asemenea face mult mai ușoară modificarea ordinii de afișare a elementelor și a relațiilor părinte-fiu între acestea;

- *O nouă arhitectură a evenimentelor.* Un alt exemplu de îmbunătățire a coerenței, este acela că toate evenimetele sunt relaționate la ascultători (eng. listener) de evenimente. Noul model de evenimente este de asemenea mai puternic, permițând evenimentelor de tip “mouse” sau “keyboard” să se propage prin multiple obiecte aflate în lista de afișare;

- *Îmbunătățirea manipulării fișierelor XML.* Un proces anterior complicat, care lucrează cu documente XML complexe este acum o plăcere cu ActionScript 3.0. Adoptând standardul E4X, ActionScript tratează acum obiectele XML într-o manieră mai familiară și inteligentă. Noua abordare permite utilizarea aceleiași sintaxe punct împreună cu obiectele relaționate prin stringuri;

- *Mai multe opțiuni de editare a textului.* Metode noi de procesare a textului permit un control complet a manipulării textelor. Se poate găsi textul dintr-o anumită linie, numărul de caractere din acea linie și chiar caracterul specificat la o anumită poziție, de exemplu, poziția indicată de pointer-ul mouse-ului. De asemenea, se poate găsi index-ul primului caracter dintr-un paragraf într-un textfield și chiar dimensiunile minime ale dreptunghiului care înconjoară un anumit caracter;

- *Noi expresii regulate.* O altă modificare a noii versiuni de ActionScript este reprezentată de noul suport pentru expresiile regulate. În loc să se manipuleze doar anumite șiruri de caractere, acum, se poate manipula textul, folosind tipuri caracter (numeric, alpha, de punctuație), elemente spațiu (spații, taburi, reveniri (eng. returns)), caractere repetitive etc.;

- *Multiple opțiuni de management a sunetului.* ActionScript 3.0 prezintă noi capabilități de management al sunetului. Practic, noua versiune conține metode de acces atât la un sunet individual cât și la toate sunetele active într-un anumit moment. Sunetele sunt poziționate în canale diferite, acest lucru ușurând interacțiunea cu multiple sunete individuale;

- *Acces nou la datele primare.* Pentru nevoi avansate, se pot accesa date binare în timpul rulării. Octeți individuali de date pot fi încărcați în timpul redării unui sunet sau manipulării unui bitmap;

- *Management nou de gestionare a domeniului de aplicație.* Într-un limbaj de programare domeniul de aplicație este folosit adeseori pentru a defini spațiul în care "trăiește" un obiect. O componentă Flash nu poate fi conținută la rândul ei decât într-o singură altă componentă. De exemplu, o componentă poate fi inclusă într-una din cele 2 componente principale ale unei aplicații, respectiva componentă fiind restricționată la părintele său direct. Structurile folosite în programare au o limită în ceea ce privește domeniul de aplicație. ActionScript 3.0 simplifică utilizarea structurilor prin urmărirea automată a acestora în timpul scrierii codului;

- *Concept îmbunătățit de programare orientată-obiect.* O ultimă îmbunătățire semnificativă a versiunii 3.0 este reprezentată de includerea claselor sigilate și a spațiilor de nume noi;

3.4.3. Merapi

Merapi este un bridge între aplicațiile AIR bazate pe Flex și Java. Bridge-ul Merapi permite comunicarea între cele 2 tehnologii prin interschimbarea de mesaje. Mesajele pot conține date complexe iar pentru serializarea lor se folosesc protocoale bazate pe AMF (Action Message Format). Java, permite dezvoltatoriilor să creeze soluții software de ultimă generație prin capabilitatea limbajului de a interacționa cu multiple aspecte al sistemului utilizatorului. Deoarece multe dispozitive externe deseori vin la pachet cu un API bazat pe Java, Merapi permite dezvoltatorului să interacționeze cu API-urile respective și să creeze aplicații care să respecte specificații pe care AIR, pe cont propriu nu ar fi capabil sa le cuprindă.

Pentru a utiliza bridgeul Merapi, atât aplicația AIR cât și aplicația Java trebuie să ruleze concomitent. Aplicația Java trebuie, de asemenea, să fie instalată separat față de aplicația AIR iar aplicația AIR nu poate instala fișierele Java. Nerespectând aceste reguli comunicarea nu poate avea loc. Nevoia copierii fișierelor Java pe o masina separată este cea ce nu permite o folosire în aplicațiile generale a bridgeului. Aplicațiile care folosesc bridgeul Merapi necesită în general să fie instalate în medii specifice. Acest lucru face ca bridgeul să fie ideal pentru aplicațiile interne.

3.4.4. JSON

JSON (JavaScript Object Notation) este un standard gratuit bazat pe text conceput pentru schimbul de date citibile. Este derivat din limbajul de scripting JavaScript pentru reprezentarea datelor structurate simple și a tablourilor asociate, numite obiecte. În ciuda relației cu JavaScript, standardul este independent de limbaj, cu interpretoare disponibile pentru majoritatea limbajelor de programare.

Tipurile de bază a JSON-ului sunt: numerele, șirurile de caractere, tipul boolean, tablourile, obiectele ca si colecții neordonate de perechi cheie:valoare) și elementul null.

3.4.5. MySQL

MySQL este cel mai popular sistem relațional de gestiune a bazelor de date. Deși este folosit foarte des împreună cu limbajul de programare PHP, folosind MySQL se pot construi aplicații în orice limbaj major. Există diverse API-uri disponibile pentru MySQL ce permit scrierea de aplicații în numeroase limbaje de programare pentru accesarea bazelor de date MySQL.

3.5. Comparație cu aplicații asemănătoare

Din multitudinea de clienți de torrente, cei mai utilizați sunt: Xunlei, µTorrent și Vuze (înainte cunoscut sub numele Azureus). Tabelul de mai jos prezintă o comparație între acești clienți, prezentând diferențele legate de: preț (licență), raportul de utilizare, data lansării, suportul pentru principalele sisteme de operare și limbajul de programare în care au fost scrise aplicațiile.

Tabel 3.1. Caracteristicile principalilor clienți de torrente

	Preț	Raport utilizare	Data lansării	Suport S.O.			Limbajul de programare
				Windows	Mac OS X	Linux	
Xunlei	Gratuit	29,30%	Anul 2006	Da	Nu	Nu	C++
µTorrent	Gratuit	25,77%	19.09.2005	Da	Da	Da	C++
Vuze	Gratuit (cu excepția dispozitivelor mobile)	24,08%	24.06.2003	Da	Da	Da	Java și Swing

În general o aplicație de acest tip comunică cu un tracker al aceluiași dezvoltator, același fiind cazul și pentru cele 3 aplicații prezentate anterior. De asemenea principalii dezvoltatori de clienți de torrente posedă și un site specializat pentru stocarea diferitelor torrente publicate pe trackerul lor și oferă posibilități mai rapide și mai bune utilizatoriilor săi să se înregistreze și să descarce torrentele dorite.

Figura 3.1. Clientul de torrente Xunlei

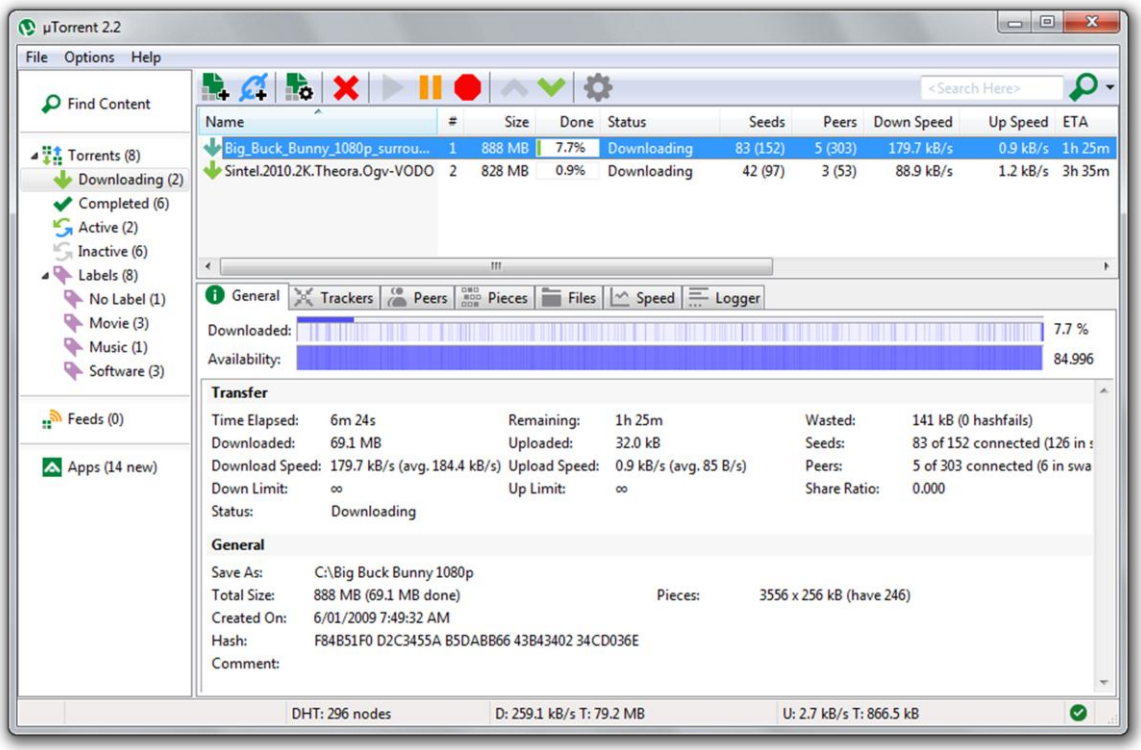

Figura 3.2. Clientul de torrente µTorrent

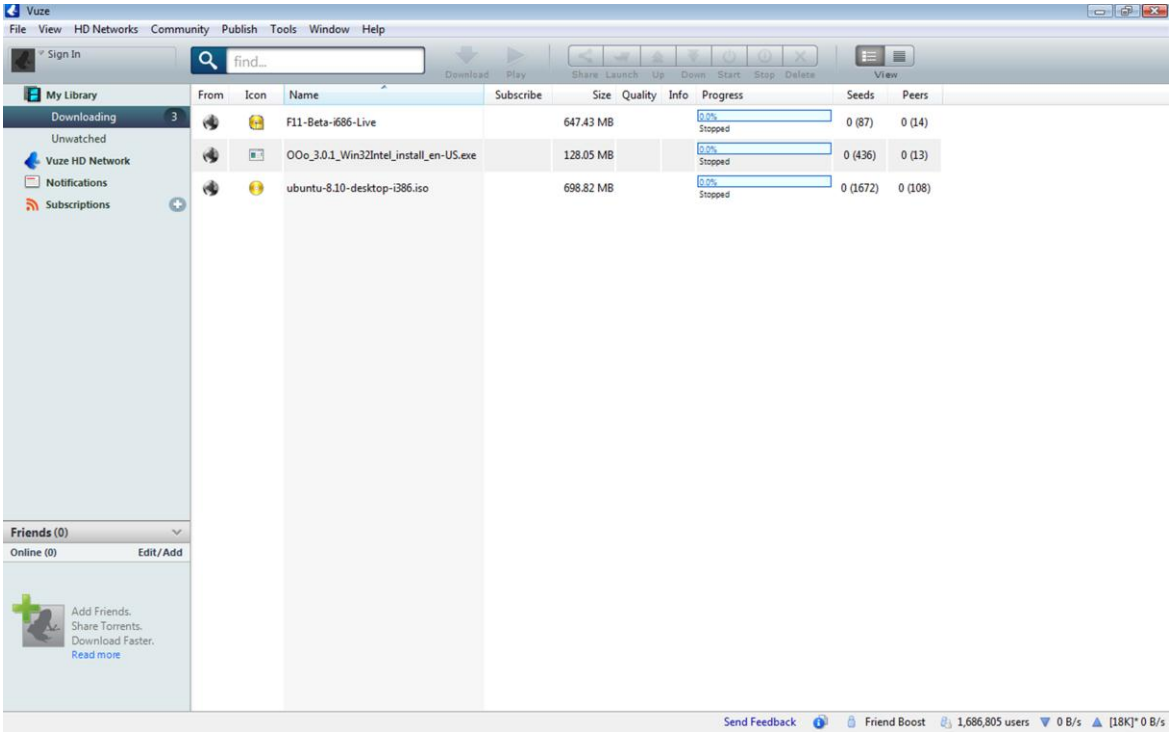

Figura 3.3. Clientul de torrente Vuze

4. ANALIZĂ ȘI PROIECTARE DE DETALIU

Deși o rețea P2P clasică este complet descentralizată, pentru o performanță mai ridicată s-a încercat o combinare a acestora cu rețelele clasice client/server. Astfel reies două tipuri de rețele P2P și anume: rețele P2P centralizate și rețele P2P descentralizate. Rețelele P2P centralizate sunt rețele P2P însă conțin și un anumit număr de servere care, în mod normal face rutarea în cadrul rețelei sau are rol de bază de date. Acestea se încadrează în categoria rețelelor P2P deoarece comunicarea între peer-uri se face strict fără intervenția serverelor, acestea din urmă ajutând doar la conectarea peer-urilor respective. În această categorie intră și sistemul prezentat în acestă lucrare. Rețelele P2P descentralizate sunt rețele P2P în interiorul cărora atât comunicarea între peeruri cât și conectarea acestora se face fără ajutorul unei părți secunde cu rol de server.

Principalii actori ai sistemului sunt clienții de torrente, între care se face concret transferul de fișiere, trackerul fiind mai degrabă o componentă secundară a acestuia. Deși o rețea clasică P2P este complet descentralizată, rutarea pachetelor în cadrul unui astfel de sistem ar dura foarte mult fără o componentă care să se ocupe special de acest lucru, datorită numărului mare de unități de calculatoare din cadrul ei. Practic, deși ocupă și el un loc într-un asemenea sistem, trackerul nu descentralizează complet sistemul, deoarece transferul concret se face stricti între peer-urile sistemului, trackerul din urmă ajutând doar la stabilirea legăturii între aceștia.

În cadrul sistemului de față trackerul este un simplu server HTTP care poate prin natura lui să răspundă la anumite cereri din partea peer-urilor, îndeplinește rol de bază de date și rutează pachetele între peer-urile rețelei pe care o gestionează. Ca și server HTTP, trackerul răspunde cererilor din partea peer-urilor din cadrul rețelei folosind protocolul HTTP.

4.1. Model abstract

În figura alăturată este prezentat un model abstract al unui sistem de transfer al fișierelor folosind torrente.

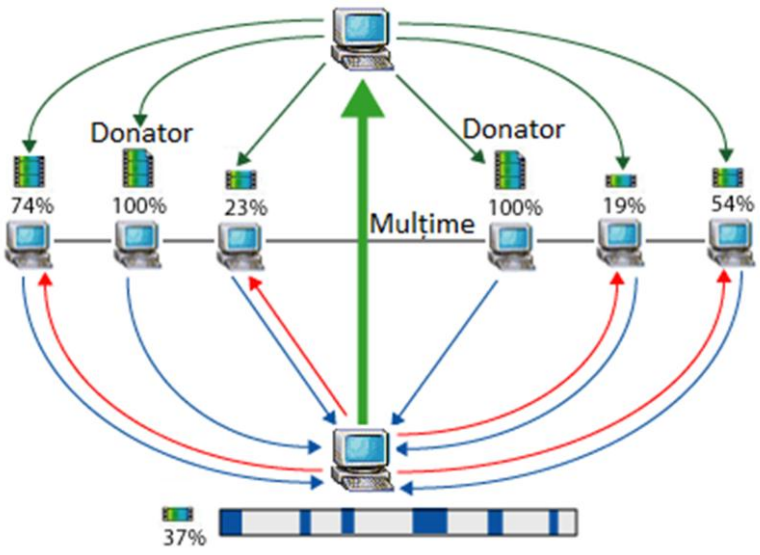

Figura 4.1. Model abstract al unui sistem de transfer al fișierelor folosind torrente

În principiu, trackerul sistemului identifică toate peer-urile mulțimii (eng. swarm) și îndrumă aplicațiile client în procesul de transfer al fișierelor între ele. Clientul, ca și aplicație recepționează și transmite mai multe pachete ale unui anumit fișier în același timp.

4.2. Protocoale utilizate

[6] Protocolul HTTP (Hypertext Transfer Protocol) este un protocol destinat utilizării în cadrul sistemelor distribuite, de colaborare și informare kypermedia. Protocolul HTTP reprezintă baza comunicării în WWW (World Wide Web).

Elaborarea standardelor HTTP a fost coordonată de către Internet Engineering Task Force (IETF) și World Wide Web Consortium (W3C), culminând cu publicarea unei serii de Cereri pentru Comnetarii (eng. Requests for Comments - RFC), în mod special RFC 2616 (iunie 1999), care definește HTTP versiunea 1.1.

În cadrul unui sistem client/server în care se folosește protocolul HTTP, un browser, de exemplu, acționează ca un client, în timp ce o aplicație care rulează pe un calculator ce găzduiește un site web funcționează ca un server. Clientul transmite un mesaj de cerere HTTP serverului, acesta din urmă, care stochează conținut sau oferă resurse, cum ar fi fișierele HTML sau îndeplinește alte funcții în numele clientului, returnează un mesaj de răspuns pentru client. Acest răspuns conține informații de stare în completarea cererii și poate conține orice informație solicitată de client în corpul mesajului său.

În majoritatea cazurilor un client este menționat ca un agent utilizator (eng. User agent). Precum și browserele web, crawler-ele web sunt reprezintă un alt tip comun de agent utilizator. Acesta din urmă includ software-le de indexare utilizate de către furnizorii de servicii de căutare. În final, browserele vocale reprezintă și ele o categorie, mai puțin comună de agenți utilizatori.

Protocolul HTTP este conceput astfel încât să permită unor elemente intermediare din cadrul unei rețele să imbunătățească sau să permită cumonicarea între client și server. Site-urile cu trafic ridicat beneficiază de cele mai multe ori de servere web cache care oferă conținut în numele serverului original, numit server de origine, pentru a îmbunătății timpul de răspuns. Serverele proxy HTTP, care se află la frontiera între două rețele, facilitează comunicarea atunci când clienții care nu dețin o adresă rutabilă la nivel global, sunt situați în rețele private prin relocarea cererilor și răspunsurilor între clienți și servere.

HTTP este un protocol care se situează în nivelul aplicație al modelului ISO/OSI conceput în cadrul framework-ului Internet Protocol Suite. Definițiile protocolului presupun un protocol de încredere în cadrul nivelului de transport al modelului de rețea, protocol pentru transferul de date între gazde. Protocolul TCP (Transfer Control Protocol) este protocolul dominant în folosință în acest caz. Cu toate acestea, protocolul HTTP, și-a găsit utilizarea chiar și cu protocoalele mai puțin sigure, cum ar fi protocolul UDP (User Datagram Protocol) sau protocolul SSDP (Simple Service Discovery Protocol).

Resursele HTTP sunt identificate și localizate în cadrul rețelei folosind identificatorii uniformi de resurse (eng. Uniform Resource Identifiers - URI) și mai precis locatorii uniformi de resurse (eng. Uniform Resource Locators - URL) folosind schemele URI http sau https. Identificatorii uniformi de resurse si HTML-ul (Hypertext Markup Language) formează un sistem intern de legare a resurselor în cadrul unei rețele, prin așa numitele documente hipertext care au dus la stabilirea WWW-ului (World Wide Web) în 1990.

Versiunea originală a HTTP (versiunea 1.0), a fost revizuită în versiunea 1.1. În timp ce versiunea 1.0 utilizează o conexiune separată la server pentru fiecare tranzacție cerere-răspuns, versiunea 1.1 poate reutiliza o conexiune de mai multe ori, pentru a prelua, de

exemplu, imagini pentru o pagină în curs de încărcare. Prin urmare, comunicarea care se realizează cu versiunea 1.1 a protocolului HTTP prezintă o latență mai mică decât stabilirea conexiunilor TCP, careprezintă o diferență majoră în raport cu acestea.

- Sesiune HTTP

O sesiune HTTP este o secvență de tranzacții cerere-răspuns în cadrul unei rețele. Inițial, un client HTTP instanțiază o cerere către server. Acesta stabilește o conexiune TCP, conexiune la un port special cu privire la o gazdă (de obicei portul 80). Un server HTTP, care ascultă pe acel port, așteaptă cereri din partea clienților iar la primirea acestora trimite înapoi o linie de start, cum ar fi "HTTP/1.1 200 OK" și un mesaj propriu, al cărui conținut reprezintă resursele cerute, un mesaj de eroare sau alte informații rezultate.

- Exemplu de sesiune HTTP

Exemplul de față prezintă o simplă conversație între un client și un server HTTP, care rulează la adresa www.example.com, portul 80.

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Figura 4.2. Cererea clientului HTTP

Cerearea clientului ,conținând în cazul de față linia de cerere si doar un singur header, este urmată de un rând liber, astfel terminându-se cu două linii noi. Header-ul HOST face distincția între numeroasele DNS-uri (Domain Name System) conținând o singură adresă IP și permițând astfel hostingul virtual bazat pe nume. În timp ce acest header este opțional în versiunea 1.0 a protocolului, este obligatoriu în versiunea 1.1.

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figura 4.3. Raspunsul serverului HTTP

Răspunsul serverului este urmat și el, la rândul lui, de o linie goală și de textul paginii cerute. Antetul/headerul Etag (Entity tag) este utilizat pentru a determina dacă o versiune a cererii recepționate din în memoria cache proprie este identică cu versiunea actuală a resurselor de pe server. Antetul Content-Type specifică tipul de Internet mass-media a datelor transmise prin mesajul HTTP, iar antetul Content-Length indică lungimea în biți a acestuia.

Serverul Web HTTP își publică abilitatea de a răspunde solicitărilor pentru anumite game octet ale documentului prin setarea antetului Accept-Ranges la valoarea „bytes”. Acest lucru este util, în cazul în care clientul trebuie să aibă numai anumite porțiuni a unei resurse trimise pe server, care este numit *“octet de servire”*. Atunci când linia Connection:close se află în headerul mesajului transmis înseamnă că serverul web va închide conexiunea TCP imediat după transmiterea acestui mesaj.

4.3. Algoritmul utilizat

Algoritmul principal al sistemului este reprezentat de modul de rutare a informațiilor în cadrul rețelei de către tracker. În principiu, trackerul folosește un serviciu numit *“TrackerService”* prin care acesta din urmă, răspunde cererilor din partea peer-urilor cu privire la informațiile despre alte peer-uri care conțin anumite torrente.

Serviciul este conținut in API-ul JBittorrent și este reprezentat printr-o clasă care moștenește clasa Service din pachetul simple.http.load.

O îmbunătățire adusă serviciu a fost acela de a ordona lista cu peer-urile care îndeplinesc criteriile din cererea inițială, după timpul necesar parcurgerii informațiilor de la sursă la destinație, astfel peer-ul care a făcut cererea face mai întâi legătura cu peer-urile care pot transmite cu o viteză mai mare informațiile necesare.

4.4. Explicații și argumentări ale soluțiilor alese

4.4.1. Java

S-a ales utilizarea limbajul de programare Java pentru realizarea backend-urilor celor două aplicații ale sistemului deoarece, în primul rând API-ul JBittorrent folosit este scris și el la rândul lui în limbajul de programare Java iar în al doilea rând pentru că este limbajul pe care îl stăpânesc cel mai bine si care prezintă cel mai bine îmbunătățiri față de alte limbaje de programare profesionale.

Din punctul meu de vedere, principalele avantaje ale acestui limbaj de programare în comparație cu alte limbaje de programare orientate obiect sunt reprezentate prin: simplitatea generală a limbajului, independența față de platforma de rulare a aplicațiilor, interpretarea limbajului, simplitatea cu care se pot creea aplicații în cadrul unei rețele și securitatea care se află în spatele limbajului.

Simplitatea limbajului Java este determinată prin proiectarea inițială a acestuia de a fi ușor de utilizat, și prin urmare, ușor de scris, compilat, depanat și de învățat față de alte limbaje de programare. Un alt motiv determinant pentru simplitatea limbajului este acela că Java folosește alocarea de memorie automată și de colectarea “gunoiului”.

Unul din cele mai importante avantaje ale limbajului Java față de alte limbaje orientate obiect este reprezentat de abilitatea sa de a trece cu ușurință de la un sistem informatic la altul. De asemenea, pentru a rula un program scris în Java este nevoie de un interpretor. Programele sunt compilate într-un cod al mașinii virtuale Java, cod numit bytecode.

Folosind Java scrierea programelor de rețea este la fel de ușoară ca scrierea și citirea de date în si dintr-un fișier. De asemenea, Java este unul din primele limbaje de programare în care s-a luat în considerare, în momentul creării, și partea de securitate. Astfel, limbajul Java, compilatorul, interpretorul și mediul de execuție s-au dezvoltat fiecare având în vedere și acest aspect.

Principalele dezavantaje ale limbajului Java sunt reprezentate prin performanța, look and feel-ul aplicațiilor a căror interfață grafică este creată cu ajutorul toolkit-ului Swing lasă deocamdată de dorit și faptul că este un limbaj cu o singură paradigmă. Aceste dezavantaje sunt din ce în ce mai nesemnificative cu fiecare versiune nouă a kit-ului de dezvoltare Java.

4.4.2. ActionScript și Flash

Interfața celor două aplicații este creată în limbajul de programare ActionScript, versiunea 3.0 împreună cu tool-ul Flash Professional de la Adobe, versiunea CS5. Practic, Flash Professional a fost folosit doar pentru crearea grafică a componentelor și funcționalitatea minimală a acestora iar ActionScript 3.0 a fost folosit pentru a implementa funcționalitate complexă interfețelor grafice.

Având în vedere asemănarea dintre limbajele de programare Java și ActionScript, acesta reprezintă principalul motiv pentru care am decis să creez interfețele celor două aplicații în acest limbaj de programare. De asemenea simplitatea în utilizare și funcționalitatea deplină a tool-ului Flash a facilitat întreaga muncă în cadrul creării interfețelor.

De asemenea, un alt motiv pentru care s-a decis utilizarea acestei tehnologii în creare interfețelor este reprezentat de gradul deplin de libertate oferit de acestea, reușind astfel să folosesc și alte cunoștințe dobândite pe parcursul studiilor de specialitate prin îmbinarea disciplinelor din categoria software, cum ar fi proiectarea interfețelor utilizator și studierea modului de interacțiune dintre om și mașină.

4.4.3. Merapi

Având în vedere diferența dintre tehnologiile folosite pentru creearea interfețelor și creearea backend-urilor am folosit un bridge special creat pentru acest scop, pentru a realiza comunicarea dintre cele două părți. Spre deosebire de comunicarea mult mai complicată folosind socketuri, Bridge-ul Merapi se ocupă singur de probleme de sincronizare a părțiilor implicate și de alte probleme de detaliu care por apărea într-o astfel de comunicare.

Practic, bridge-ul conține două librării, câte una pentru fiecare tehnologie, care conțin metode speciale de transmitere și recepționare a mesajelor.

4.4.4. JSON

JSON este un format foarte simplu de înțeles și folosit de reprezentare și interschimbare a datelor între diferite aplicații software. Formatul este unul textual, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri complexe de date și este în special folosit pentru transmiterea datelor structurate în cadrul unei rețele, procesul purtând numele de serializare.

De asemenea, format este disponibil în toate limbajele de programare importante prin existența unei multitudini de librării.

4.4.5. MySQL și „connection pooling”

În primă fază, API-ul folosit pentru crearea acestui sistem folosea ca sistem de gestiunea a datelor necesare lucrul cu fișiere XML. Având în vedere cantitatea de informații care poate să fie memorată de un singur tracker în cadrul unui asemenea sistem, am considerat mult mai profesional, utilizarea unui sistem profesional de gestiune a unei baze de date. Astfel, am folosit MySQL pe partea bazei de date iar pentru o performanță ridicată am folosit și o tehnică relativ noua și anume “connection pooling”.

Tehnica de connection pool presupune existența unor conexiuni deja create în momentul primirii unei cereri către baza de date reducând astfel timpul necesar interogării

acesteia. Utilizarea unui connection pool presupune memorarea unor conexiuni create, însă având în vedere cantitatea de memorie medie în prezent, este recomandabil să se renunțe la o minimă de memorie în favoare unui timp mult mai scurt. În cadrul acestui proiect am testat cele trei sisteme de gestiune a unei baza de date(XML, MySQL fără connection pool și MySQL cu connection pool), testele rezultate urmând a fi prezentate în capitolul 6. "Testare și validare".

4.5. Arhitectura conceptuală a sistemului

Cele două componente principale din diagrama de componente reprezintă cei doi actori ai sistemului. Diagrama prezintă principalele componente interne ale actoriilor și tehnologiile folosite în creearea lor, precum și protocoalele de comunicare între acestea.

Pentru componenta "peer" interfața a fost creată folosind limbajul de programare ActionScript, versiunea 3.0, mediul de rulare a acesteia fiind AIR de la Adobe, un mediu special creat pentru rularea aplicațiilor de tip desktop. Backend-ul componentei este realizat folosind limbajul de programare Java iar în această sub-componentă nu am intervenit cu mari modificări din punctul de vedere al structurii. Am decis însă să separ cei doi conectori ai componentei (PeerConnector și TrackerConnector) deoarece consider că sunt componente principale ale întregului sistem și aici se pot aduce în viitor îmbunătățiri semnificative.

Comunicarea între interfață și backend este realizată prin bridge-ul Merapi iar ca principiu de serializare s-a folosit formatul JSON. De asemenea, comunicarea între componentele principale ale sistemului se face folosind protocolul HTTP.

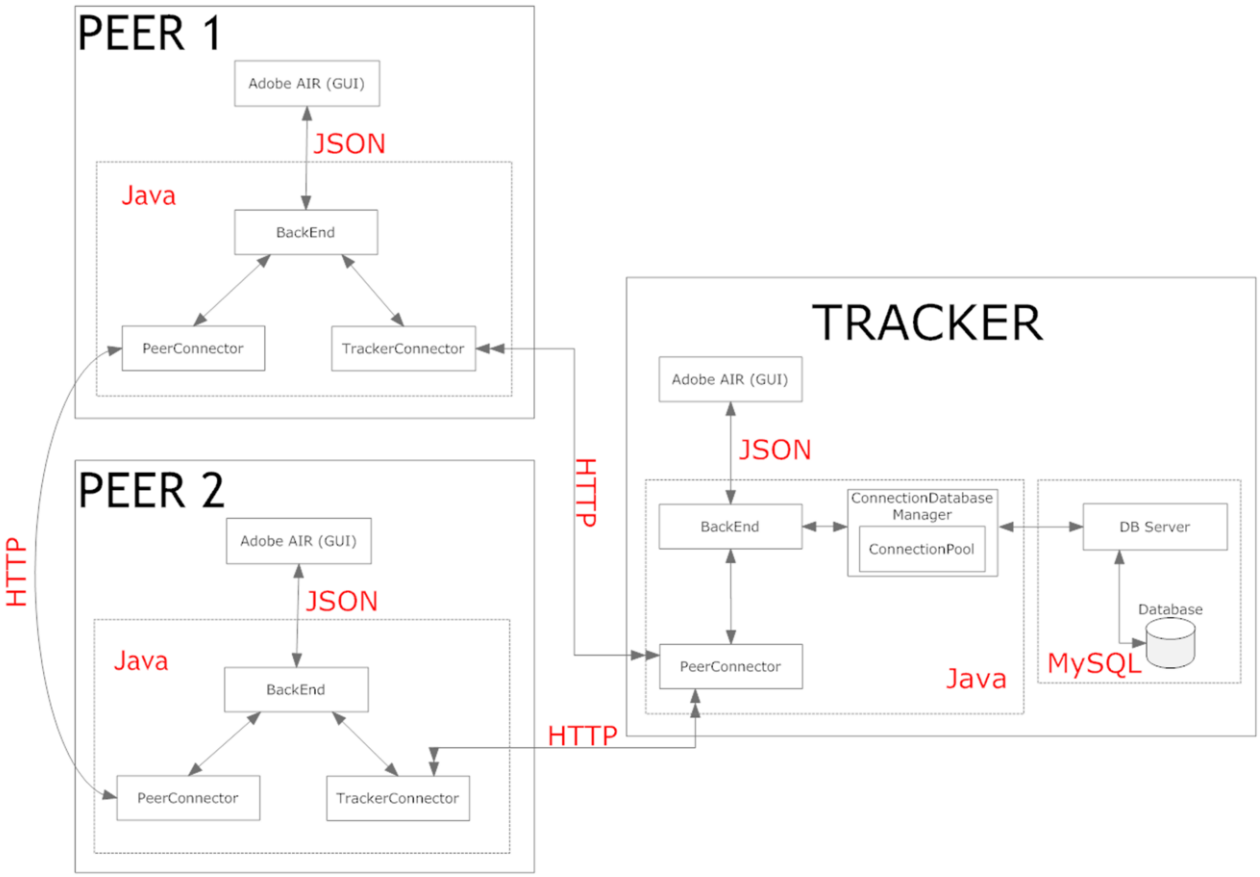

Figura 4.4. Arhitectura conceptuală a sistemului

Componenta “tracker” este puțin mai complexă din punctul de vedere al structurii componentelor interne prin utilizarea unui număr mai mare de tehnologii și prin complexitatea rolului si acțiuniilor ei în sistem.

Astfel, atât interfața cât și backend-ul sunt asemănătoare cu cele ale componentei peer din punctul de vedere al tehnologiilor folosite și al structurii sub-componentelor ei.

Spre deosebire de componenta peer, componenta tracker conține un server de baze de date, baza de date asociată și un manager care face legătura între backend-ul componentei și baza de date respectivă și conține connection pool-ul utilizat. Serverul si baza de date sunt realizate folosind tehnologia MySQL iar managerul este realizat folosind limbajul de programare Java.

5. IMPLEMENTARE

În acest capitol se va prezenta în detaliu implementarea celor două aplicații desktop, aplicația client și aplicația tracker. Se vor face câteva considerații generale asupra design-ului aplicațiilor, după care se vor descrie în profunzime, insistându-se pe elementele cele mai importante.

Pentru dezvoltarea backend-urilor celor două aplicații s-a folosit IDE-ul Eclipse, unul din cele mai complete și profesionale tool-uri la ora actuală pentru crearea aplicațiilor în Java.

Pentru crearea interfețelor s-a folosit în primul rând platforma Flash Professional de la Adobe, versiunea CS5, pentru crearea grafică a componentelor interfețelor utilizator pentru cele două aplicații și integrarea acestora în aplicațiile finale. Utilizând acest tool s-a putut implementa fără probleme funcționalitatea minimală necesară fiecărei componente.

De asemenea, s-a folosit editorul FlashDevelop pentru crearea aplicațiilor corespunzătoare interfețelor utilizator a celor două aplicații. Datorită complexității și rolului interfețelor utilizator în cadrul celor două aplicații, s-a decis structurarea acestora într-un mod cât mai profesional si modular posibil.

Pentru crearea bazei de date și a elementelor relaționate cu aceasta s-a utilizat tool-ul MySQL Workbench datorită completitudinii de servici pe care le pune la dispoziția dezvoltatorilor de acest tip de aplicații.

Deoarece, interfețele utilizator a celor două aplicații ocupă un rol important în structura întregului sistem s-a decis utilizarea a două design patern-uri: paternul MVC (Model View Controller) și patternul Singleton. Voi prezenta în continuare principalele aspecte legate de aceste pattern-uri.

5.1. Design patern-uri folosite

5.1.1. Model View Controller

Conform [8], Model View Controller (MVC) reprezintă o arhitectură software, considerat în prezent un pattern arhitectural folosit în ingineria software.

În [9] se specifică faptul că patternul este unul comportamental care se situează în mod obișnuit la nivelul de componentă în cadrul arhitecturii unui sistem. Scopul modelului este acela de a împărți o componentă sau un sub-sistem în trei părți logice separate: modelul, view-ul și controller-ul, acest lucru ușurând posibilitatea modificării fiecărei părți în parte.

Pettern-ul MVC este folosit în general în cadrul unor aplicații complexe ce prezintă un număr mare de date către utilizator. În acest caz se preferă separarea datelor (modelului) de interfața cu utilizatorul (view), astfel încât modificarea interfeței să nu afecteze datele, iar datele să poată fi organizate fără a se modifica interfața cu utilizatorul. Această problemă este rezolvată prin introducerea componentei intermediare și anume controller-ul.

Modelul – este domeniul specific reprezentării informațiilor asupra cărora operează aplicația. Numeroase aplicații folosesc mecanisme de reținere persistente, precum bazele de date.

View – realizează interfața cu utilizatorul, prezentând modelul într-o formă accesibilă.

Controller-ul – procesează și răspunde la evenimente (acțiunile utilizatorului) și invocă schimbări în model.

Astfel, utilizatorul va interacționa cu interfața într-un anumit mod (apăsând un buton), iar controller-ul va procesa evenimentul de intrare de la interfață și va accesa și updata modelul. Componenta view va accesa modelul pentru a genera o interfață, iar componenta interfață va aștepta noi interacțiuni ale utilizatorului.

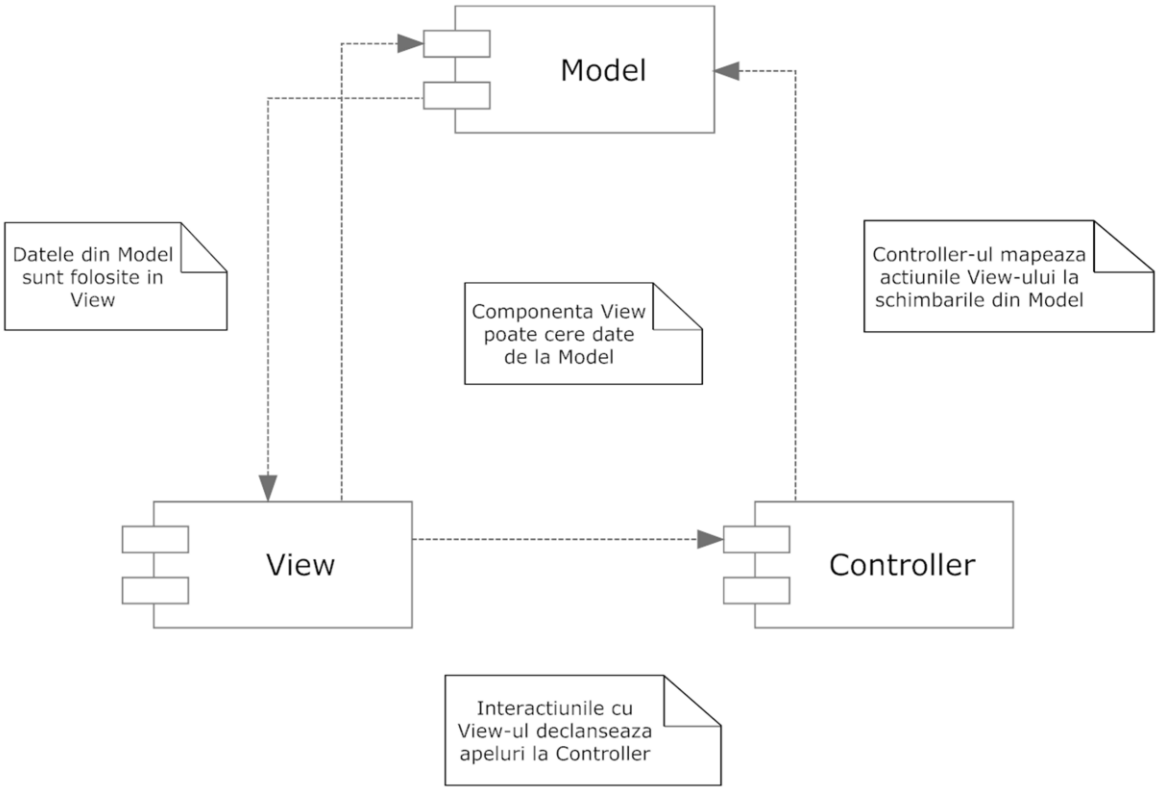

Figura 5.1. Diagrama de componente MVC

5.1.2. Singleton

Conform [9], design pattern-ul Singleton este un pattern creațional care se folosețte la nivel de obiect. Scopul utilizării acestui design pattern este acela de a avea o singură instanță a unei clase în întreg sistemul, permițând în același timp celorlalte clase să acceseze instanța fără nici un impediment.

Design pattern-ul Singleton asigură crearea unei singure instanțe de către mașina virtuală Java. Pentru a asigura controlul asupra instanțierii, constructorul clasei respective trebuie să fie privat. În acest moment apare o problemă: posibilitatea creării unei instanțe, iar această problemă se rezolvă prin crearea unei metode statice (denumite în general getInstance()). Metoda crează instanța, doar în cazul în care nu a fost creată în prealabil, și returnează referința clasei respective apelatorului metodei inițiale.

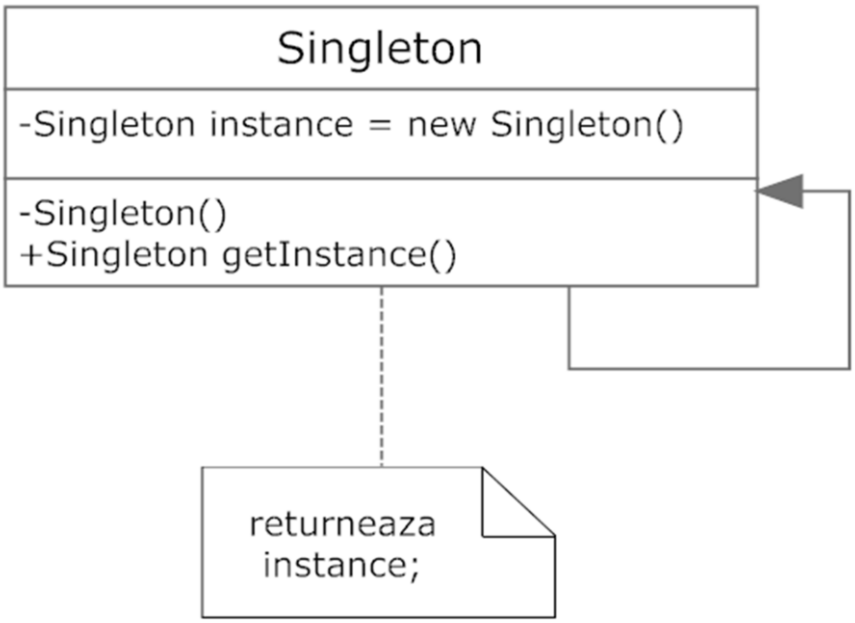

Figura 5.2. Diagrama de clase Singleton

5.2. Diagrama de pachete

Din [10] reiese faptul că diagrama de pachete are rol de a prezenta dependințele dintre pachetele unui sistem. Pe lângă standardele UML reprezentate de diagramele de pachete, acestea mai prezintă două tipuri de dependințe:

- Importarea pachetelor;
- Îmbinarea pachetelor.

Importarea unui pachet reprezintă o relație între numele de spațiu importat și un pachet, indicând faptul că numele de spațiu importat adaugă numele membrilor pachetului respectiv în spațiul de nume propriu.

Îmbinarea pachetelor reprezintă o relație directă între două pachete, relație care indică faptul că, conținutul celor două pachete este combinat. Acest procedeu este similar procesului de generalizare în sensul că, elementul sursă adaugă în mod conceptual caracteristicile elementului țintă în propriile caracteristici, rezultând un element terț care cuprinde toate caracteristicile.

Diagrama de pachete prezentată în figura 5.3. reprezintă un model general, fiind specific atât aplicației client cât și aplicației tracker. Având în vedere faptul ca backend-ul aplicațiilor nu conține în mod specific o interfață, strucutra diagramei de pachete pentru această sub-componentă este mai simplă. Pachetele prezente aici sunt: client, data, control si jBittorrentAPI.

Pachetul client conține clasa main a aplicației, pachetul data conține toate tipurile de date, în special JSON, utilizate în cadrul comunicării cu interfața, pachetul control conține handler-ii sub-componentei iar pachetul jBittorrentAPI conține sursele librăriei client al API-ului Jaba Bittorrent.

De cealaltă parte, pentru sub-componenta GUI, având în mod implicit o componentă interfață, s-a utilizat pattern-ul MVC pentru o mai bună modularizare și pentru ușurarea aducerii de modificări ulterior.

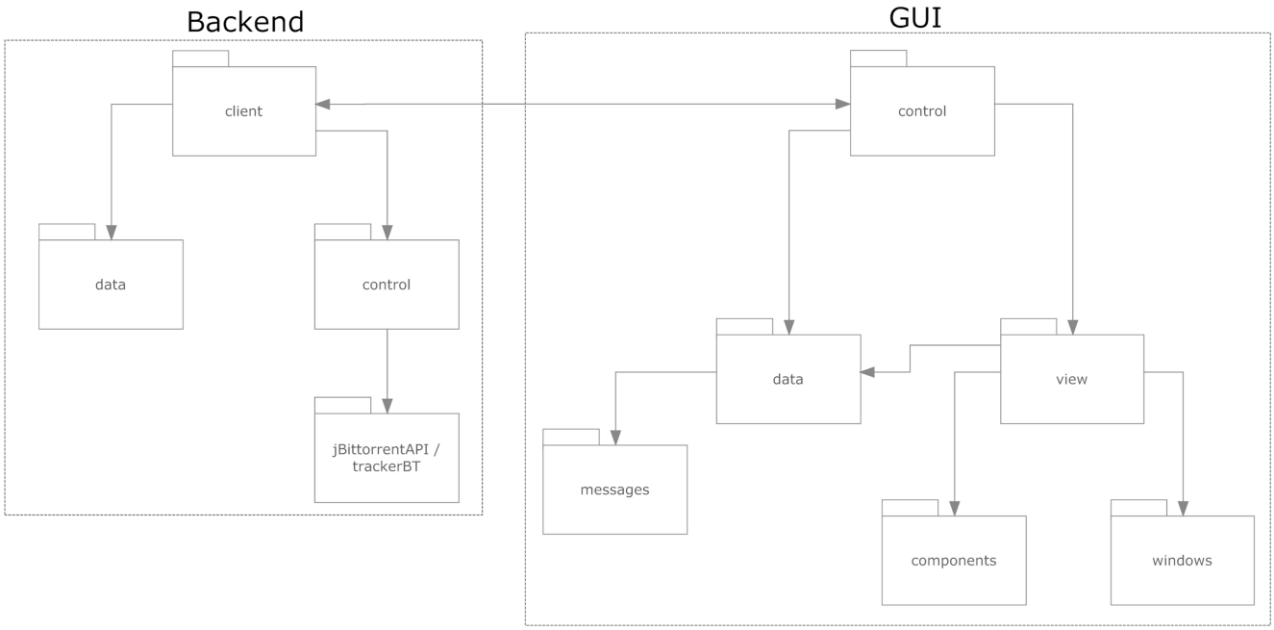

Figura 5.3. Diagrama generală de pachete

5.3. Diagrame de secvențiere

În continuare vor fi prezentate într-un mod mai detaliat principalele acțiuni posibile din partea utilizatorului: crearea unui nou torrent, publicarea unui torrent pe un tracker activ și descărcarea unor fișier cu ajutorul unui torrent. Acțiunile prezentate nu sunt specifice unei singure aplicații, fluxul acestora interacționând atât cu clientul cât și cu tracker-ul.

Pentru o prezentare mai detaliată a principalelor acțiuni s-au folosit diagrame de secvențe. Diagramele de secvențe prezintă interacțiunea între procesele care i-au parte la executarea unei acțiuni. Printr-o diagramă de secvențe se prezintă diferitele procese sau obiecte care "trăiesc" simultan și mesajele interschimbate între acestea. Utilizând astfel de diagrame se poate prezenta specificare unui simplu scenariu într-o manieră grafică.

Într-o astfel de diagramă, perioada de existență a proceselor sau a obiectelor este reprezentată prin linii paralele verticale iar schimbul de mesaje între acestea prin săgeți orizontale.

5.3.1. Creare torrent

Crearea unui torrent reprezintă una din principalele acțiuni posibil de realizat cu ajutorul sistemului de față. Acțiunea începe prin adresarea unei comenzi din partea utilizatorului, aplicației client. Acest lucru se face în mod grafic, prin apăsarea butonului de creare, după ce în prealabil utilizatorul a completat toate câmpurile existente în fereastra de creare a unui torrent.

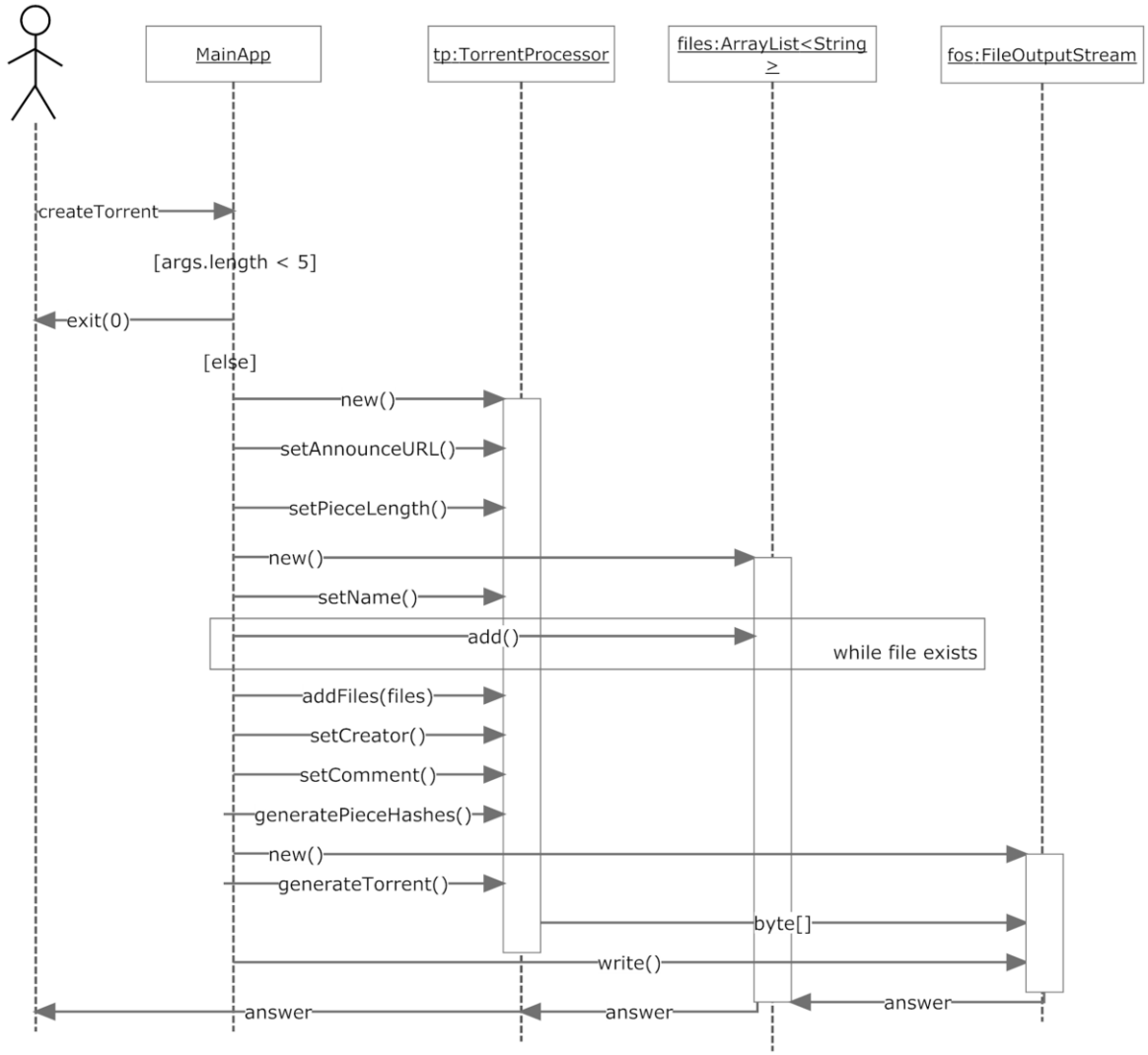

Figura 5.4. Diagramă de secvențiere – Creare torrent

După activarea comenzii de creare a unui torrent, se verifică existența tuturor parametriilor necesari, inexistența acestora ducând la terminarea imediată a rulării aplicației. În cazul în care numărul parametriilor este corect, se crează un obiect de tipul TorrentProcessor care conține atributele specifice unui fișier .torrent. După setarea acestor parametrii se crează fișierul propriu-zis, în final urmând a se transmite un răspuns utilizatorului printr-un mesaj de notificare.

5.3.2. Publicare torrent

Publicarea unui torrent necesită două elemente: torrentul care urmează a fi publicat și trackerul pe care urmează să se publice torrentul respectiv să fie activ. După introducerea informațiilor necesare publicării unui torrent în interfața utilizator și executarea comenzii respective, clasa principală a aplicației client face un apel unui obiect de tipul ConnectionManager care publică torrentul specificat prin calea acestuia în sistemul în care se află.

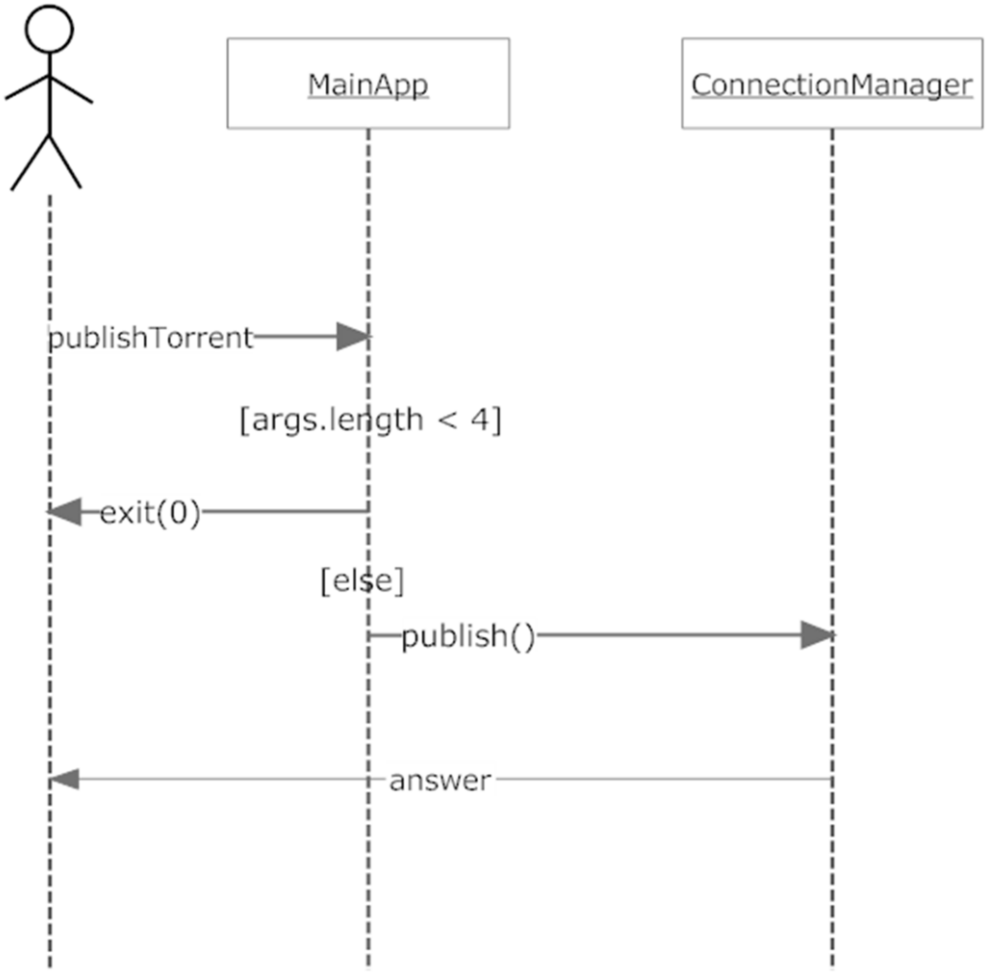

Figura 5.5. Diagramă de secvențiere – Publicare torrent

După încercarea de publicare a torrentului, obiectul ConnectionManager returnează un răspuns prin care se notifică utilizatorul dacă torrentul selectat a putut sau nu să fie publicat.

5.3.3. Descărcare fișiere

Descărcarea fișierelor reprezintă activitatea de bază a unui sistem de transfer al fișierelor folosind torrente. Practic, descărcarea fișierelor reprezintă transferul de fișiere specificate într-un anumit torrent între doi sau mai mulți utilizatori ai unui sistem de acest tip în cadrul aceleiași rețele.

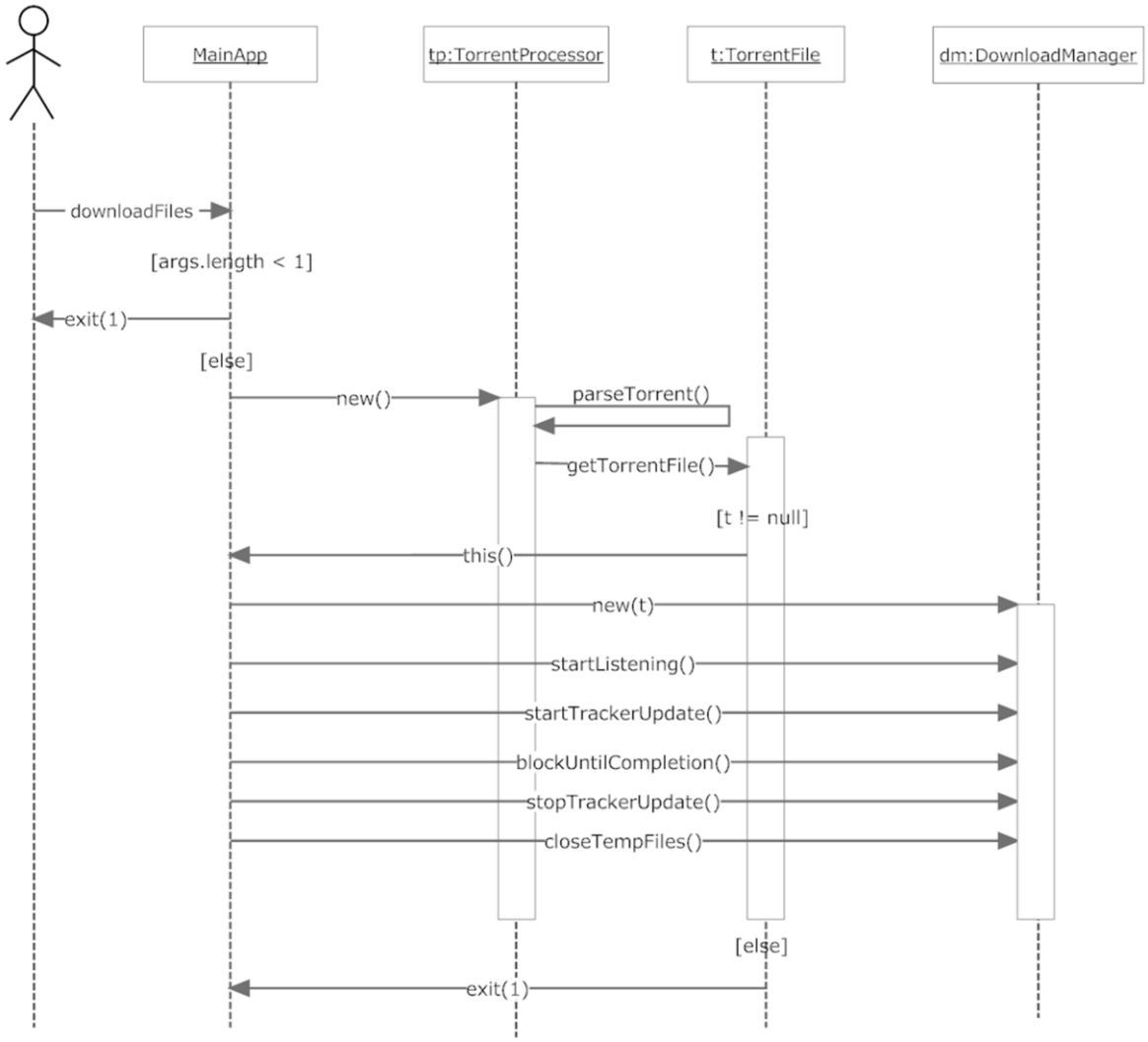

Figura 5.6. Diagramă de secvențiere – Descărcare fișiere

5.4. Prezentarea GUI-ului aplicației client

Luând în considerare faptul că s-a utilizat API-ul Java Bittorrent care include o mare parte a backend-urilor celor două aplicații, interfața aplicațiilor cu utilizatorul a fost creată în întregime folosind tehnologii diferite față de cele utilizate în crearea backend-urilor.

În procesul de creare a interfețelor utilizator s-au folosit diferite practici studiate de-a lungul studiilor academice în legătură cu acest domeniu. S-a încercat crearea unei interfețe plăcute și ușor de utilizat, astfel, s-au folosit tehnologii care oferă un grad maxim de libertate atât din punct de vedere grafic cât și comportamental.

Rolul important pe care îl are interfața, cea a aplicației client în mod special, în cadrul întregului sistem se datorează faptului că aceasta reprezintă principala diferență dintre aplicația client și alți clienți de torrente.

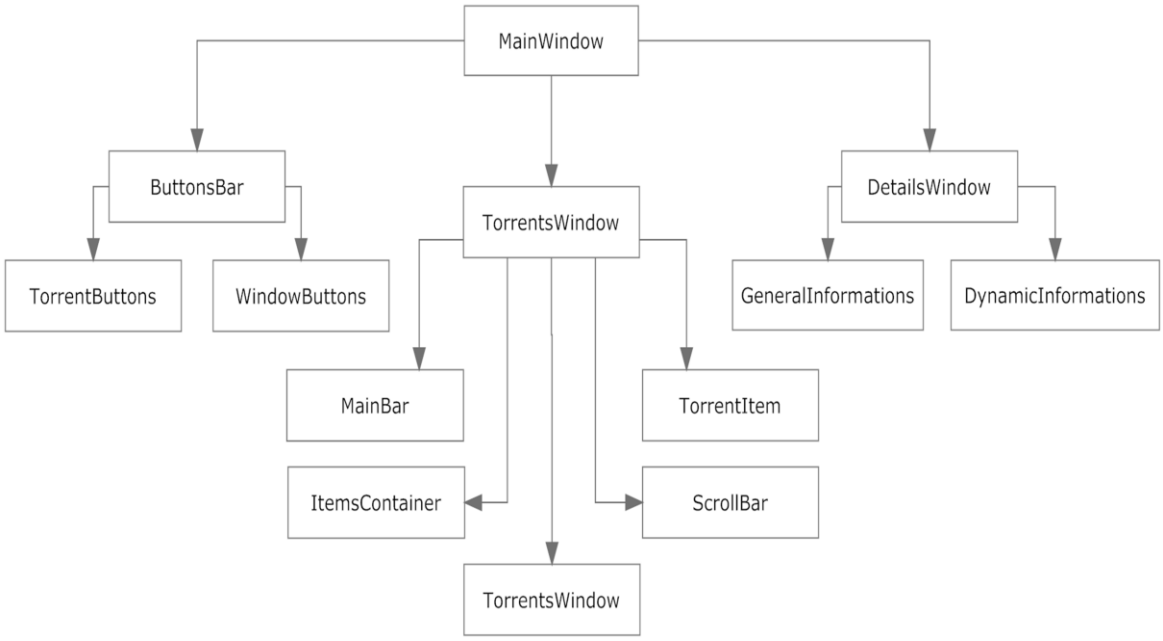

Figura 5.7. Structura interfeței utilizator pentru aplicația client

Interfața aplicației client este bazată pe structura prezentată în figura 5.7. Fereastra principală este împărțită în trei zone: o zonă de butoane, fereastra torrentelor active și fereastra de informații a torrentului selectat.

Zona de butoane este și ea împărțită în două categorii: butoanele relaționate cu torrentele (creare torrent, publicare torrent, sharing torrent, căutare torrent și butonul de setări) și butoanele relaționate cu fereastra aplicației (butonul de minimizare și butonul de închidere).

Fereastra torrentelor active conține diferite componente grafice, unel dintre ele fiind dinamice. Componenta "TorrentsWindow" a acestei ferestre a rol strict grafic. Această componentă este utilizată pentru a adăuga un efect de umbră zonei respective.

Celelalte componente ale ferestrei torrentelor active sunt: "MainBar", "ScrollBar", "ItemsContainer", "TorrentItem". "MainBar" reprezintă bara statică în care se află numele categoriilor de informații ale torrentelor active. "ScrollBar" reprezintă bara de scroll necesară în cazul în care numărul torrentelor active devine prea mare pentru a putea fi vizulizate în mod normal. "TorrentItem" reprezintă elementul grafic corespondent unui torrent activ în cadrul ferestrei torrentelor active. Acest element este dinamic, în el afișiindu-se numele și progresul torrentului respectiv, numărul de parteneri și donatori disponibili momentan și vitezele de încărcare, respectiv descărcare.

Ultima componentă a ferestrei principale este reprezentată de fereastra de detalii. În această componentă se afișează atât informațiile generale ale torrentului selectat în fereastra torrentelor active, cât și diferite informații dinamice prezente și în fereastra torrentelor active.

Structura componentelor interfeței utilizator, în cazul aplicației client este una arborescentă. Avantajul principal este acela că se pot structura chiar și cele mai simple componente iar dezavantajul că modificarea unei componente care conține alte sub-componente ar putea să implice și modificarea sub-componentelor ei.

5.5. Utilizarea bridgeului Merapi

Comunicarea între interfața utilizator și backend-ul aplicației se face folosind un bridge special realizat pentru acest scop, atât pentru aplicația client cât și pentru aplicația tracker. Bridge-ul Merapi conține două librării, una pentru Java iar cealaltă pentru ActionScript, librării care la rândul lor conțin clase specializate în comunicarea între cele două tehnologii, ascunzând față de utilizator detalii asupra comunicării (portul folosit pentru comunicare, protocolul folosit, sincronizarea etc.)

În cazul interfeței utilizator unde s-a folosit tehnologie ActionScript, bridgel Merapi conține o librărie .swc denumită *„merapi-core-flex.swc”*. După introducerea în aplicație a librăriei respective, se folosește obiectul Singleton *Bridge* a cărui apel al metodei de transmitere a unui mesaj este ilustrat în imaginea de mai jos. De asemenea, pentru recepționarea unui mesaj din backend, prin aceași instanță singleton al bridgeului trebuie inregistrat un handler care implementează interfața *IMessageHandler* și implicit metoda *handleMessage*.

```
//transmiterea unui mesaj
Bridge.getInstance().sendMessage(new Message('message', data));
...
//inregistrarea handler-ului de mesaje
Bridge.getInstance().registerMessageHandler('message', MessageHandler);
```

Figura 5.8. Utilizarea librăriei merapi-core-flex.swc

După cum se poate vedea și în figura anterioară, mesajele transmise prin acest bridge conțin două câmpuri: un string cu rol de identificator al unui canal de comunicație și mesajul propriu-zis.

Pentru backend, bridge-ul Merapi conține o librărie .jar numită „merapi-core-0.1.8-beta.jar”. În acest punct am întâlnit cea mai notificabilă problemă în cazul utilizării brideg-ului și realizării comunicării între cele două tehnologii. Nespecificându-se, în documentația produsului, faptul că la rândul ei, librăria „merapi-core-0.1.8-beta.jar” are nevoie de alte librării pentru a putea fi utilizată s-a avut nevoie de un proces de analiză a modului de funcționare a librăriei respective, descoperiind în final librăriile necesare: log4j-1.2.15.jar, flex-messaging-common.jar, flex-messaging-core.jar, spring.jar și tools.jar.

Utilizarea librăriei „merapi-core-0.1.8-beta.jar” este asemănătoare utilizării librăriei merapi-core-flex.swc, singura diferență fiind aceea că handler-ul pentru mesajele recepționate trebuie să extindă clasa *MessageHandler* pe când în interfața utilizator librării corespunzătoare trebuie să implementeze interfața *ImessageHandler*.

5.6. Utilizarea formatului JSON

Având în vedere faptul că tipul datelor transmise între interfață și backend nu se rezumă doar la șiruri de caractere am avut nevoie de un format ușor de înțeles și simplu de implementat pentru datele transmise între cele două componente. Astfel am folosit formatul JSON. Utilizarea acest formatului a fost foarte simplă de realizat datorită multitudinii de tehnologii pentru care există librării specializate de acest tip.

În pricipiu formatul JSON se utilizează pentru comunicarea de date pe Internet, dar are anumite caracteristici moștenite care limitează eficiența utilizării lui cu acest scop. Cele mai multe dintre limitările formatului, sunt în general legate de formatul datelor textuale. De

asemenea aceste limitări se aplică și altor formate cum ar fi XML și YAML. De exemplu, în ciuda faptului că, în mod normal, sunt generate de un algoritm, parsarea trebuie realizată caracter cu caracter. În plus, standardul nu are nici o resursă pentru compresia datelor, memorarea singulară a fiecărui string diferit sau referințe la anumite obiecte. Compresia poate să fie aplicată datelor JSON formatate însă datele rezultate în urma decompresiei ar trebui din nou parsate caracter cu caracter pentru recunoașterea cuvintelor-cheie a tag-urilor si delimitatorilor.

Singura problemă notificabilă pe care am intâmpinat-o a fost aceea de a defini un set de reguli de comunicare între cele două tehnologii. Astfel am creat două clase speciale, câte una pentru fiecare tehnologie, în care am memorat cheile perechilor care sunt transmise în cadrul comunicării între cele două tehnologii.

Astfel am împărțit cheile în trei mari categorii: câmpurile generale reprezintă cheile care se folosesc la cel mai înalt nivel în structura unui mesaj JSON, câmpurile secundare reprezintă cheile care sunt folosite în structurile interioare ale mesajului JSON, iar a 3-a categorie reprezintă numele diferitelor metode care se pot alfa în mesajul JSON cu scopul apelării din tehnologia diferită.

```
{"houseNumber":"742",
  "street":"Evergreen Terrace",
  "city":"Springfield",
  "postcode":"49007",
  "country":"USA",
  "comments": ["Deliveries accepted.","Familiar address, huh?",""]
 }
```

Figura 5.9. Exemplu de mesaj JSON

5.7. Structuri de date

Rolul principal al bazei de date a fost de a crește performanța tracker-ului de la un anumit număr de utilizatori în sus. Baza de date este specifică tracker-ului, aceste din urmă fiind singurul care memorează datele necesare funcționării sistemului în cadrul rețelei din care face parte.

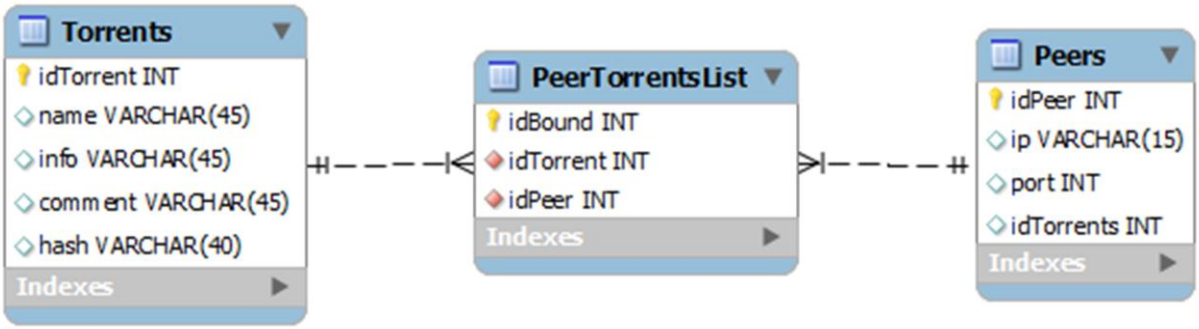

Figura 5.10. Diagrama bazei de date

Baza de date, este din puctul de vedere al structurii una foarte simplă. Practic, s-au mapat tabelele XML folosite de API-up Java Bittorrent folosit și s-a creat un tabel de legătura între un peer și torrentele deținute el.

Tabele bazei de date sunt următoarele: Torrents, PeerTorrentsList și Peers. Tabelul Torrents este folosit pentru a memora informațiile torrentelor publicate pe tracker-ul care deține baza de date respectivă, pentru a putea oferi informații cu privire la acestea celorlalte perr-uri. Informațiile memorate în acest tabel sunt: idTorrent (câmp care reprezintă ID-ul torrentului), name (câmp folosit pentru memorarea numele torrentului), info (câmp folosit pentru memorarea informațiilor adiționale despre torrent), comment (câmp folosit pentru memorarea commentariului adăugat de peer-ul care a publicat torrentul), hash (câmp folosit pentru memorarea cheii de criptare a torrentului pe tracker).

Tabelul Peers este folosit pentru a memora informațiile despre peer-rurile care publică diferite torrente pe tracker. Câmpurile tabelului sunt: idPeer (câmp folosit pentru memorarea ID-urilor peer-urilor), ip (câmp folosit pentru memorarea IP-ului), port (câmp folosit pentru memorarea portului folosit pentru a comunica cu trackerul curent) și idTorrents (câmp folosit pentru a face legătura între peer-ul respectiv și torrent-ele relaționate cu acesta).

Tabelul PeerTorrentsList este folosit pentru a face legătura între un peer și torrent-ele relaționate cu acesta. Cele trei câmpuri ale tabelului sunt: idBound (câmp care reprezintă ID-ul unei legături), idTorrent (câmp folosit pentru memorarea ID-ului unui torrent) și idPeer (câmp folosit pentru memorarea ID-ului unui peer).

5.8. Connection pool

Conform [11] un “connection pool” reprezintă o colecție de sesiuni deja conectate la baza de date prin intermediul cărora se poate reduce timpul necesar stabilirii unei conexiuni la baza de date. Connection pool-ul poate fi furnizat prin diferite metode, în funcție de software-ul utilizat. Probabil, cea mai bună modalitate este de a utiliza un connection pool este de a utiliza software-ul de pe partea serverului deoarece acesta funcționează pentru toate tipurile de conexiune, nu doar pentru conexiunile specifice unui software anume.

În interiorul managerului bazei de date, connection pool-ul conține în primă fază o colecție de conexiuni nerealizate. În momentul când este nevoie de o conexiune suplimentară la baza de date, o nouă conexiune se realizează, aceasta din urmă rămânând activă până la oprirea aplicației tracker. Astfel, în primă fază doar utilizatorul pentru care trebuie să se realizeze o conexiune nouă va trebui să aștepte timpul necesar creării respectivei conexiuni.

O modalitate de eliminare a acestui timp, este aceea de a realiza toate conexiunile de la bun început, acest lucru necesitând însă memorie suplimentară. Având în vedere cantitatea de memorie medie din zilele noastre și memoria necesară realizării unei conexiuni la o bază de date, această metodă din urmă începe să fie utilizată din ce în ce mai des.

După cum se va prezenta în capitolul 6. Testare și validare, utilizarea unui connection pool îmbunătățește performanța trackerului într-un mod semnificativ.

6. TESTARE ȘI VALIDARE

6.1. Testarea sistemului

Pentru a crea un sistem funcțional care respectă toate cerințele impuse am folosit următorul plan de testare format din următorii pași:

- Pasul 1 – testarea funcțională a metodelor, după terminarea scrierii lor;
- Pasul 2 – testarea combinată a diferitelor părți a celor două aplicații;
- Pasul 3 – testarea generală a întregii aplicații (numită și testare black-box).

6.1.1. Testarea funcțională a metodelor după scrierea lor

Acestă etapă reprezintă prima etapă în cadrul procesului de testare și a început imediat după scrierea metodelor.

După crearea fiecărei metode, aceasta a fost testată prin rularea ei pe baza unor date de intrare aleatorii. De exemplu, pentru interfețele utilizator a celor două aplicații am testat metodele de inițializare a componentelor grafice pentru fiecare pagină în parte.

Rezultatele obținute au fost comparate cu design-ul inițial al interfețelor celor două aplicații, ajutând apoi la modificare acestora până la obținerea unui grad ridicat de satisfacție din partea utilizatorului.

6.1.2. Testare combinată a mai multor părți

În această etapă au fost testate diferite funcționalități a aplicație care cuprind mai multe componente terminate. De exemplu, neincluzând interfața utilizator propriu-zisă, am testat comunicarea între logica de business a interfeței și backendul aplicației pentru principalele acțiuni ale aplicației client.

Verificarea s-a făcut prin verificarea, de exemplu, a unui torrent creat, comparând datele incluse în el cu datele de intrare folosite în testare.

6.1.3. Testarea generală

După terminarea implementării, s-au testat ambele aplicații, luând în considerare toate cazurile de utilizare create inițial. În primă fază, testarea a decurs fără a întâmpina dificultăți majore, cele întâlnite fiind ușor de rezolvat. După corectarea greșelilor întâlnite, aplicațiile au fost testate din nou pentru a asigura o calitate ridicată sistemului rezultat.

În continuare se prezintă un scenariu de testare generală a aplicației client:

- Utilizatorul apasă butonul de creare a unui torrent nou;
- Utilizatorul introduce date în câmpurile ferestrei de creare a torrentului, lăsând câteva câmpuri necompletate;

- Utilizatorul apasă butonul de creare a torrentului iar în mod așteptat acesta este atenționat că nu a completat unul sau mai multe câmpuri obligatorii;
- Utilizatorul completează câmpurile lăsate anterior goale și apasă din nou pe butonul de creare a torrentului, de acestă dată torrentul creeându-se în mod normal și conținând datele corecte.

6.2. Testarea îmbunătățirilor aduse

Având în vedere materialele necesare anumitor testări ale îmbunătățiriilor aduse, cum ar fi, testarea impactului ordonării listei de peer-uri care conțin un anumit torrent nu am reușit să testez doar acele feature-uri care pot fi testate local.

Astfel am verificat diferențele, din punctul de vedere al timpilor, între utilizarea fișierelor XML, a unei baze de date simple și a unei baze de date cu un connection pool. Testele au fost făcute pe elemente relativ egale, atât fișierele cât și tabelele având același număr de câmpuri și același număr de înregistrări.

Rezultatele testelor pot fi văzute în tabelul de mai jos.

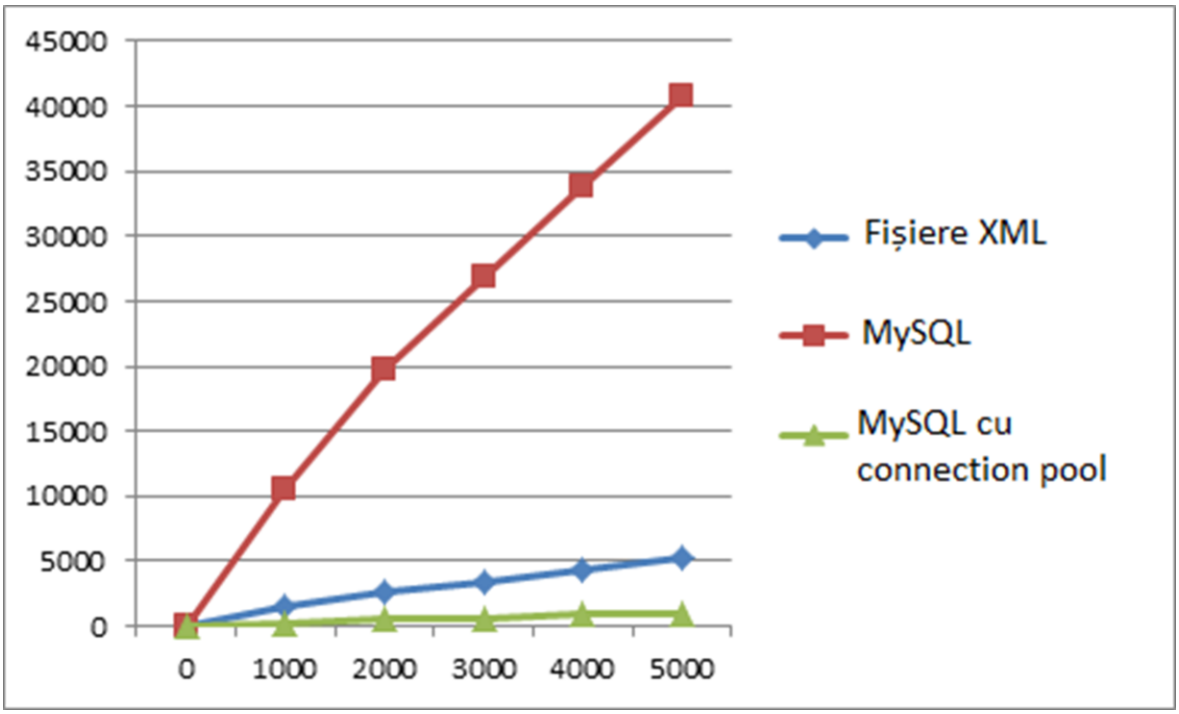

Figura 6.1 Rezultatele testului timpilor de acces la baza de date

Din rezultatele testului reiese faptul că, folosind o librărie Java specializată în lucrul cu fișiere XML (JDOM) este mai avantajos decât utilizarea unei baze de date MySQL simple, din punctul de vedere al timpilor necesari diferitelor interogări. Acest lucru se explică prin faptul că, timpul necesar creării și inițializării unei conexiuni la baza de date necesită un timp îndelungat.

Pentru a îmbunătății timpii necesari interogării bazei de date MySQL am folosit un connection pool în care se stochează o mulțime de conexiuni la baza de date, conexiuni gata create și inițializate.

Rezultatele testului sunt: 0.3 secunde necesare pentru o parcurgere completă la fiecare mie de înregistrări, 20 de secunde sunt necesare pentru aceași cantitate de informații în cazul bazei de date MySQL simple și 0.06 secunde sunt necesare în cazul unei baze de date MySQL căreia ăi este asociată un connection pool.

7. MANUAL DE UTILIZARE

Punerea în funcțiune a sistemului presupune rularea ambelor aplicații existente în componența sa: clientul și trackerul. Atât clientul cât și trackerul vor putea fi rulate doar prin copierea arhivelor si deschiderea lor.

Utilizarea aplicației client presupune în primă fază existența mediului de rulare AIR de la Adobe instalat pe mașina pe care se dorește rularea respectivă. De asemenea, este necesară și existența mediului de rulare Java (JRE). Ambele kit-uri de instalare a acestor tehnologii sunt gratuite și vor putea fi găsite pe site-ul producătorilor lor.

Față de aplicația client, aplicația tracker necesită și existența unui server pentru baze de date MySQL. Acesta din urmă este și el gratuit și poate fi descărcat de pe site-ul producătorului său.

Datorită importanței semnificativ ridicate a aplicației client în cadrul sistemului s-a decis explicarea doar a acțiunilor principale care pot fi executate pe aceasta. Astfel, după deschiderea aplicației utilizatorul poate să creeze un torrent, să publice un torrent pe un tracker activ, să descarce fișiere de la ceilalți utilizatori ai aplicației în cadrul aceleiași rețele și să modifice diferiții parametri de rulare a aplicației.

- Crearea unui torrent nou

În cazul în care utilizatorul dorește să creeze un torrent nou, acesta va trebui să apese în primă fază pe butonul corespunzător acestei acțiuni. După apăsarea butonului respectiv, se va deschide fereastra de creare a unui torrent nou (figura 7.1.) utilizatorul urmând să introducă informațiile necesare creării torrentului. Acest lucru se face poate face foarte ușor prin simpla introducere a informațiilor necesare și adăugarea fișierelor dorite.

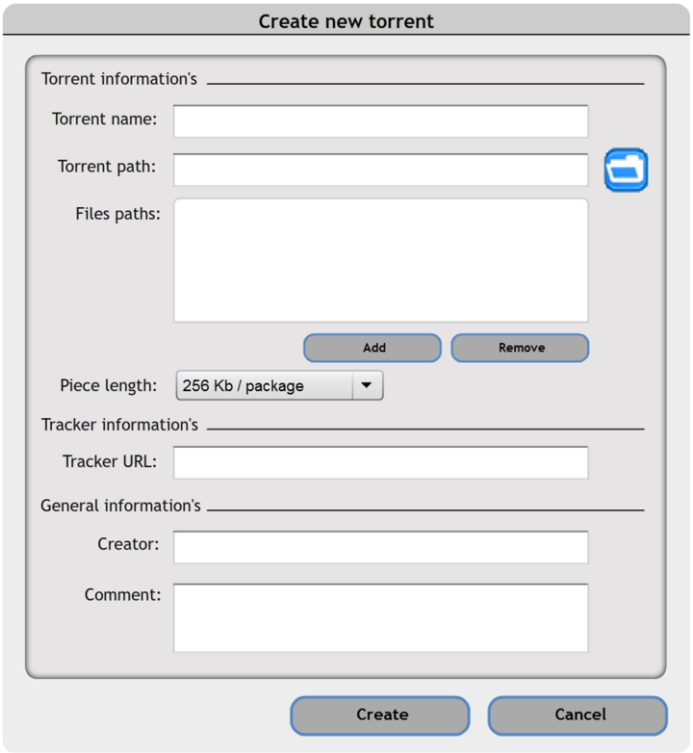

Figura 7.1. Fereastra de creare a unui torrent nou

După introducerea tuturor informțiilor necesare, utilizatorul va trebui să apese butonul de creare a torrentului pentru a putea finaliza acțiunea. După apăsarea butonului respectiv, datele introduse de utilizator vor fi transmise la backend pentru a fi procesate, iar în cazul în care sunt corecte se va crea torrentul dorit inițial.

Unul dintre avantajele utilizării acestui sistem este că utilizatorul primește constant feedback din partea aplicației prin diferite mesaje prezentate în ferestre pop-up. Un exemplu ar fi, cazul în care un torrent este creat cu succes, caz în care utilizatorul este anunțat prin mesajul din figura 7.2.

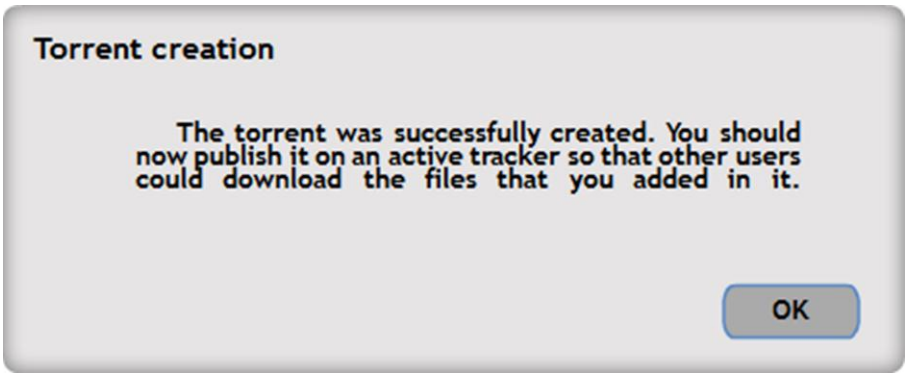

Figura 7.2. Mesajul de confirmare a creării unui torrent nou

Feedback-ul din partea aplicației vine și în cazul în care torrentul nu a putut fi creat din diferite cauze (probleme de tracker, date introduse greșit etc.).

- Publicarea unui torrent

Publicarea unui torrent se face doar după ce în prealabil acesta a fost creat.Operația este asemănătoare cu cea de creare a torrentului, utilizatorul urmând a executa un șir de pași asemănător cu cel din cazul creării torrentului.

La fel ca în cazul creării torrentului, utilizatorul este informat dacă operația a decurs cu succes sau nu.

- Descărcarea fișierelor folosind un torrent

Pentru a porni descărcarea datelor descrise într-un torrent, utilizatorul va trebui să execute operația de deschidere a unui torrent. În cazul în care structura torrentului nu este corectă utilizatorul va primi un mesaj de atenționare. În cazul în care aplicația client nu a întâmpinat probleme în deschiderea torrentului, se va crea un nou element în fereastra torrentelor active asemănător figurii 7.3.

Din această zonă, utilizatorul va putea să interacționeze directe asupra torrentelor active pentru a opri descărcarea fișierelor descrise într-un anumit torrent, pentru a șterge un torrent descărcat în totalitate sau chiar pentru a reîncărca informațiile legate de partenerii și donatorii unui torrent activ.

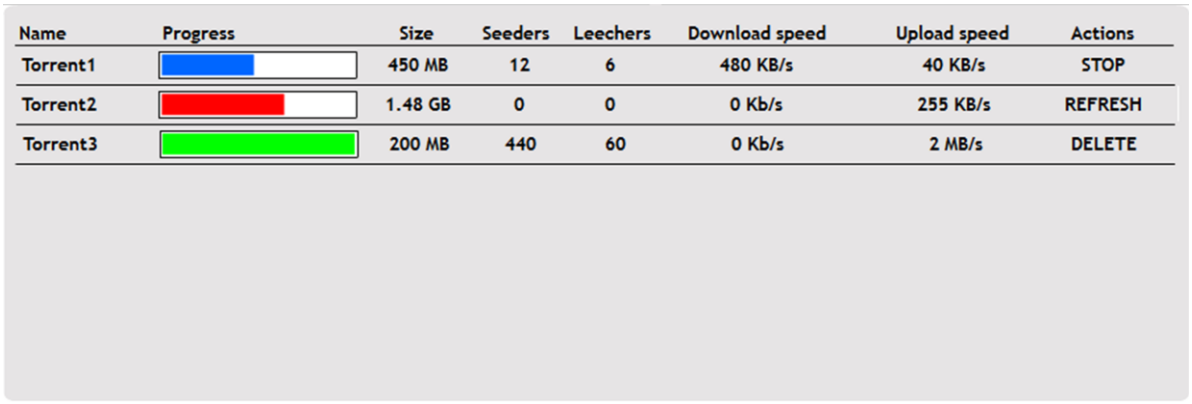

Figura 7.3. Fereastra torrentelor active

În momentul în care un torrent devine activ, utilizatorul are opțiunea de a vizualiza informații detaliate despre torrentul respectiv. Aceste informații sunt de două tipuri: informații statice (informații generale despre torrentul selectat) și informații dinamice legate de stadiul în care se află procesul de descărcare a torrentului selectat. Figura 7.4. prezinta un model al ferestrei de informații descris anterior.

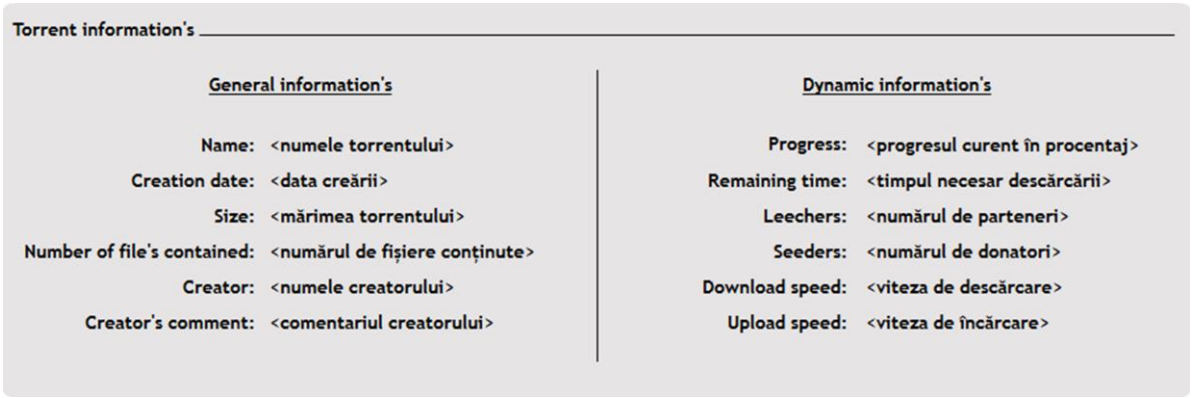

Figura 7.4. Fereastra de informații a aplicației client

- Schimbarea parametriilor aplicației

Pe lângă operațiile descrise mai sus, utilizatorul are posibilitatea de a modifica diferiți parametrii specifici aplicației client. Pentru a face acest lucru, utilizatorul trebuie să apese butonul de deschiderea a ferestrei cu parametrii și să modifice setările dorite.

Prin cele mai semnificative posibilități din acest punct de vedere sunt utlizarea unui proxy prin intermediul căruia aplicația să poată comunica cu un anumit tracker, utilizarea

unor URL-urile specifice unui anumit tracker pentru a scuti utilizatorul de a introduce aceleași informații în mod repetat în cazul creării și publicării de torrente și utilizarea unor căi standard pentru locația fișierelor în curs de descărcare, la fel pentru a scuti utilizatorul de a fi nevoit să introducă aceste informații într-un mod repetat.

8. CONCLUZII

8.1. Realizări

Sistemul realizat în cadrul acestui proiect reprezintă o alternativă la transferul fișierelor în cadrul unei rețele folosind torrente.

Au fost implementate cerințele specificate în capitolul 2. S-a dezvoltat un sistem simplu de transfer a informațiilor digitale în cadrul unei rețele de calculatoare folosind fișiere speciale numite torrente. S-a îndeplinit obiectivul principal, acela de a încerca parcurgerea tuturor pașilor de creare a unui produs software într-un mod cât mai profesional cu putință. De asemenea s-a îndeplinit și obiectivul secundar al proiectului, acela de a include cât mai multe tehnologii posibil.

Pentru corectitudinea rulării sistemului, s-au creat două aplicații diferite: o aplicație cu rol de client, care poate descărca și încărca diferite informații cu alți utilizatori din cadrul aceleiași rețele și o aplicație cu rol de tracker, prin intermediul căreia se face legătura între actorii principali ai sistemului.

S-a reușit găsirea unor tehnologii și componente existente deja, ușurând astfel munca și utilizând diferitele componente cu scopul pentru care au fost create. S-a dedus modul de utilizare a diferitelor librării folosite și s-a reușit integrarea acestora în cele două aplicații.

8.2. Dezvoltări ulterioare

Datorită procesului de dezvoltare continuu în care se află domeniul rețelelor P2P, posibilitatea dezvoltăriilor ulterioare este vastă.

În primul rând, se pot îmbunătății interfețele celor două aplicații, folosind tehnologii actualizate și mai bune din punct de vedere al performanței. Un exemplu ar fi utilizarea componentelor Flex în loc de Flash, această tehnologie fiind mult mai ofertantă din punctul de vedere al dezvoltatorului.

În al 2-lea rând, Java fiind un limbaj open-source, dezvoltatorii își aduc contribuția în mod permanent în depistarea și remedierea erorilor și chiar găsirea de noi soluții în cele mai diversificate domenii, oferind astfel posibilitatea de a îmbunătați funcționalitatea părții de backend atât pentru aplicația client cât și pentru cea de tracking.

Pe termen scurt îmi propun realizarea următoarelor obiective suplimentare celor dezvoltate până în acest moment:

- Notificarea clienților activi cu privire la publicarea diferitelor torrente publicate pe un anumit tracker;
- Dezvoltarea bazei de date din aplicația tracker pentru a putea stoca informații suplimentare atât despre torrente cât și despre clienți sistemului;
- Adăugarea unei funcționalități de înscriere și logare a clienților înainte de a publica diferite torrente pe un tracker, pentru o securitate sporită;
- Implementarea unei funcționalități prin care clienți pot evalua torrentele folosite sau calitatea acestora;

- Depistarea și eliminarea automată a torrentelor structurate greșit;
- Implementarea unui sistem de urmărire a fișierelor descărcate cu aplicația client pentru a împiedica schimbarea locațiilor acestora și împiedica astfel posibilitatea de descărcare de către alți utilizatori.

9. BIBLIOGRAFIE

[1] Andrew Stuart Tanenbaum, *"Computer Networks"*, Prentice Hall, 4th Edition, 17 Martie 2003.

[2] Karl Wiegers, *"Software Requirements"*, Microsoft Press, 2nd Edition, 2003.

[3] Baptiste Dubuis, Martin Rajman, David Portabella Clotet, *"Java Bittorrent API"*, École Polytechnique Fédérale de Lausanne, 2007.

[4] Ștefan Tănasă, Cristian Olaru, Ștefan Andrei, *"Java de la 0 la expert"*, Polirom, Ediția a 2-a, 2003.

[5] Rich Shupe, Zevan Rosser, *"Learning ActionScript 3.0"*, O'Reilly, 2008.

[6] Wikipedia Inc., "Hypertext Transfer Protocol", 2011, http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[7] Wikipedia Inc., "BitTorrent (protocol)", 2011, http://en.wikipedia.org/wiki/BitTorrent_(protocol)

[8] Wikipedia Inc., „Model-view-controller", 2011, http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[9] Stephen Steling, Olav Maassen, "Applied Java Patterns", Prentice Hall PTR, 1st Edition, 1 Decembrie 2001

[10] Wikipedia Inc., "Package diagram", 2011, http://en.wikipedia.org/wiki/Package_diagram

[11] Simon Riggs, Hannu Krosing, "PostgreSQL 9 Administration Cookbook", Packt Publishing, 2010

[12] Wikipedia Inc., "Use case", 2011, http://en.wikipedia.org/wiki/Use_case

[13] Robert Cecil Martin, "UML for Java Programmers", Prentice Hall, 2002

[14] Craig Larman, „Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", Addison Wesley Professional, 3rd Edition, 20 Octombrie 2004

Glosar de termeni

API – Application Programming Interface

Byte code - Seturi de instrucțiui proiectate pentru execuția eficientă a interpretoarelor software

DNS - Domain Name System

Donator (Seeder) - Un peer care are datele 100% complete și le poate împărți

FTP - File Transfer Protocol

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

IDE - Integrated Development Environment

IEEE - Institute of Electrical and Electronics Engineers

IP - Internet Protocol

JRE – Java Runtime Environment

Mesaj pop-up – O fereastră copil care necesită interacțiunea cu utilizatorul, înainte ca acesta să se poată întoarce la aplicația părinte

MVC – Model View Controller

P2P - Peer-to-Peer

Partener (Leecher) – Un peer sau client care nu are datele 100% complete

Singleton - Design pattern folosit pentru restricționarea instanțierii unei clase la un singur obiect

SSDP - Simple Service Discovery Protocol

swc - Pachet de simboluri Flash și cod ActionScript precompilate

TCP - Transmission Control Protocol

Torrent - Protocol pentru descărcare de fișiere

Tracker - Server care asistă comunicarea între peer-uri

UDP - User Datagram Protocol

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

WWW - World Wide Web

Lista figurilor