Sviluppo di un prototipo di File System Distribuito:

P2PFileSystem

Project Manager:
(R)

Team:
(R)
(R)

Indice:

1. Introduzione

1.1 Obiettivo

Questo documento ha il compito di illustrare le specifiche funzionali e progettuali del software distribuito P2PFileSystem. In seguito verranno quindi illustrate sia le funzioni sviluppate che i requisiti che tale applicazione deve soddisfare. La descrizione sarà, per scelta progettuale, sintetica ma esaustiva.

1.2 Campo di Validità

Il nome del software è P2PFileSystem. P2PFileSystem è un'applicazione distribuita open source che permette di condividere file attraverso una rete locale.
Nello specifico, questo software può funzionare sia in modalità “Server” che in modalità “Client”, a discrezione dell'utente. La parte Server non fa altro che immagazzinare una lista di file (i loro nomi) da condividere proveniente da ogni nodo della sua sottorete, la parte Client, invece, sfrutterà le funzioni del server per indicare i file da mettere in condivisione e accedere ai file remoti condivisi da altri nodi.

P2PFileSystem può essere utilizzato in ogni rete locale su cui sia possibile effettuare chiamate broadcast (oppure in reti wan, conoscendo l’indirizzo ip del server) e sulle cui macchine siano installate piattaforme Java/Rmi e MySql.

1.3 Definizioni, Acronimi e Abbreviazioni

Vengono di seguito spiegati alcuni termini che verranno utilizzati in seguito, per evitare eventuali ambiguità di interpretazione dei tali in sede di lettura del documento:

Server – nodo del sistema su cui è in esecuzione P2PFileSystem in modalità “Server”. Questa macchina permette ad altri nodi di connettersi a lei per condividere i propri file ponendosi in ascolto su una porta specifica.

Client - nodo del sistema su cui è in esecuzione P2PFileSystem in modalità “Client”. Il client richiede i servizi del Server a seconda delle esigenze dell'utente. La sua unica funzionalità è quindi quella di interfacciarsi col Server per condividere file e accedere a eventuali file remoti.

DB – Abbreviazione per Data Base. Un data base è un archivio strutturato in modo tale da consentire la gestione dei dati attraverso interazioni software.

Client mittente – nodo del sistema che viene contattato dal Server centralizzato in quanto è arrivata una richiesta per uno dei suoi file

Client ricevente – nodo del sistema che richiede l'accesso a un file remoto e ne fa richiesta prima al Server, poi al client mittente proprietario del nodo.

1.4 Sommario

Nella seconda sezione di questo documento sono elencate le funzioni a disposizione dei possessori di questo software e eventuali vincoli da rispettare nel momento dell'utilizzo.

La terza sezione descrive i requisiti che l'applicazione deve soddisfare, sia funzionali che non funzionali. I requisiti sono stati concordati in fase progettuale considerando la natura del software da sviluppare e le specifiche richieste da parte del committente.

In appendice A è visibile la topologia del sistema e un esempio di file di configurazione.

2. Descrizione Generale

2.1 Contesto

Il pacchetto applicativo può essere usato da qualsiasi utente privato o organizzazione che intenda sfruttare funzionalità di file-sharing remoto all'interno di una propria rete locale composta da più nodi interconnessi.
La natura semi-centralizzata della gestione dei file remoti rende questo software particolarmente adatto a reti di piccole dimensioni con buoni requisiti di stabilità di connessione e sicurezza interna.

All'interno del pacchetto applicativo saranno comprese funzionalità di gestione di un DB locale (creato e ospitato dalla macchina Server) con la capacità di mantenere riferimenti ai file condivisi nella rete. In particolare verranno sfruttate le interrogazioni per ricavare gli attributi dei file al momento della richiesta di condivisione.
L'interazione con il server sarà possibile solo sfruttando l'applicativo lato client, che dovrà quindi essere in esecuzione sul nodo.

2.2 Funzioni

Le macrofunzioni che il pacchetto mette a disposizione sono:

- Gestione della ricerca di server nella rete locale.
- Accesso e autenticazione su un server.
- Gestione del DB dei file messi in condivisione e degli utenti connessi.

- Gestione dell'accesso remoto a file:
    - ricerca del file nel database
    - contatto del client mittente
    - condivisione della locazione del client mittente con il client ricevente
    - download del file
    - cancellazione del file
    - upload del file

- Condivisione di un file
    - scelta del file
    - salvataggio del file nella lista centralizzata

- Disconnessione da un server

2.3 Caratteristiche Utente

Un utilizzatore dell'applicazione P2PFileSystem può decidere di farlo in modalità Client o Server.
In entrambi i casi, le capacità medie richieste sono minime, in quanto ogni operazione è guidata da un'interfaccia grafica intuitiva per gli utenti dei sistemi moderni.
L'unica scelta che si è chiamati ad effettuare è quella tra l'esecuzione lato Client o lato Server, ma anche questa non risulta difficoltosa, sempre per merito della interfaccia.

2.4 Vincoli Generali

Per poter sfruttare le funzionalità dell'applicazione è necessario che questa venga eseguita su una macchina su cui risiede una versione di Java Development Kit 1.6 (o successive), in quanto P2PFileSystem è completamente scritta in linguaggio Java.
Come facilmente intuibile, ogni macchina deve potersi collegare in rete. Di fondamentale importanza è la possibilità di avere, sulla stessa macchina, i permessi di interrogare (per esempio in broadcast) la rete stessa. Eventuali Firewall potrebbero creare problemi nella comunicazione; è quindi necessario accettare datagrammi UDP sulla porta definita e connessioni TCP/IP per la comunicazione tra client e server.

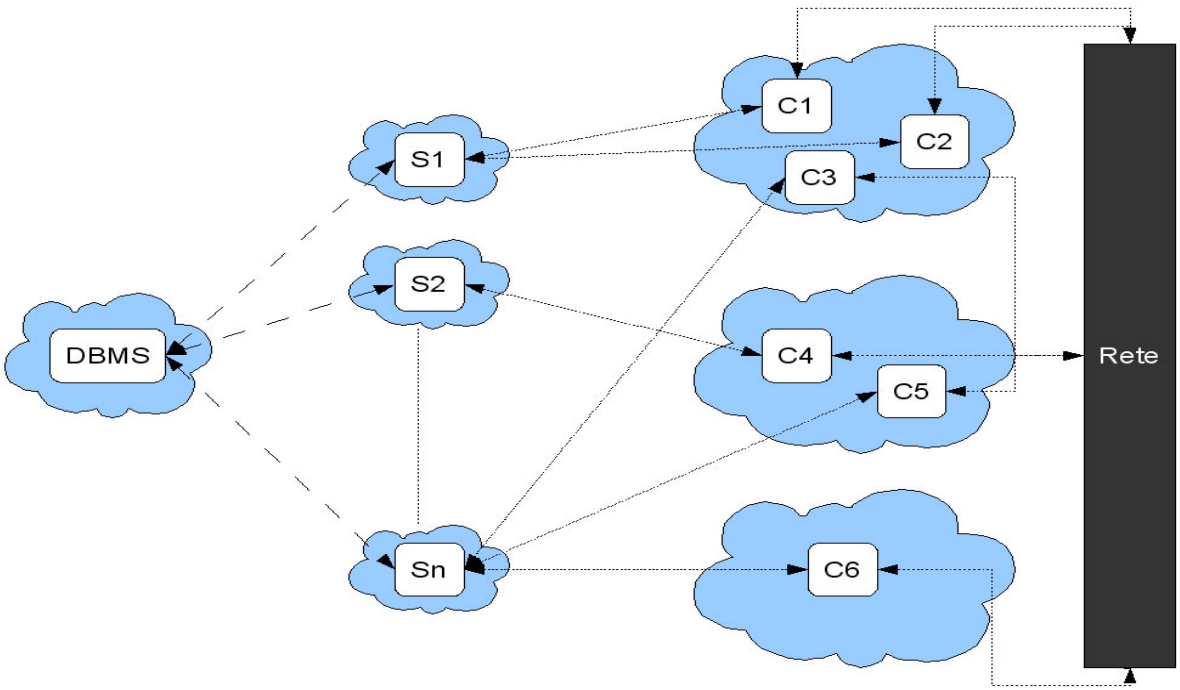
2.5 Assunzioni e Dipendenze

Non sono necessarie conoscenze particolari per l'uso di questo software, se non quelle basilari di un sistema software attraverso un'interfaccia grafica intuitiva. Requisiti di questo tipo sono riscontrabili nella maggior parte degli utenti moderni.

2.6 Requisiti da analizzare in futuro

Replicazione dell'applicativo server:

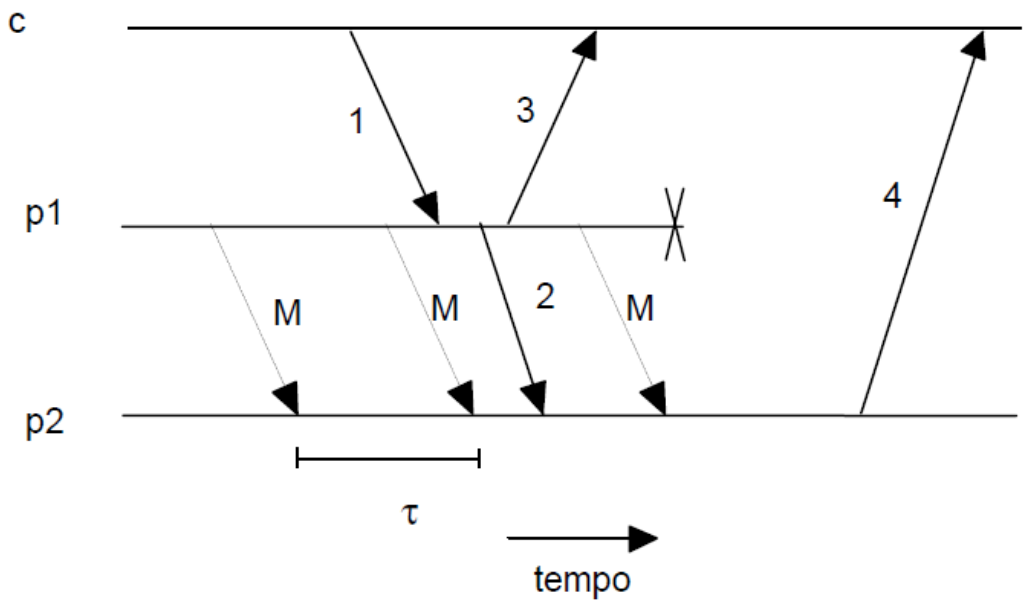
Siccome la tecnologia JDBC unita al DBMS MySQL permettono l'interrogazione di un DB anche da macchine remote, specificando come unico parametro, a differenza della chiamata classica, l'indirizzo del nodo sul quale è in esecuzione il servizio MySQL, questo processo può essere distaccato dall'esecuzione del lato server del programma P2PFileSystem.
Mediante banali modifiche del programma, è possibile che più server accedano allo stesso DBMS, implementando in modo semplice, ma funzionale, la loro replicazione.



Come si può vedere dalla figura, tutti i client (C) connessi ai vari server (S) che condividono lo stesso DBMS, possono comunicare tra loro pur essendo su reti completamente distaccate.

Replicazione dell'applicativo DBMS:

Dopo la replicazione dell'applicativo server, l'unico collo di bottiglia del sistema (server), risulta essere il DBMS, la cui replicazione può essere altresì veloce e trasparente. Il meccanismo necessita semplicemente di due query SQL e di un algoritmo di switching in caso di guasti, come ad esempio quello visibile in figura.



Replicazione dei contenuti:

Un tipo di replicazione dei file sottile, ma valida, potrebbe essere implementata nel seguente modo:

- Ogni file è identificato (oltre che dai classici attributi) da un hash del suo contenuto (alta univocità).
- Un client che richiede una replicazione di un suo file, lo invia (con metodi già implementati) ad un altro client, scelto da [TBD] .
- Il client riceve il file e automaticamente lo inserisce tra le sue condivisioni. In questo caso è necessaria, oltre alla già implementata funzione di regolazione delle connessioni multiple, l'implementazione di una politica di controllo, come potrebbe essere quella della quota disco.
- Il file "replicato" è individuabile in caso di ricerca rispetto all'originale, solo dal suo hash. La consistenza di tale file non può essere garantita in caso di modifica a meno di implementare anche un meccanismo che la preveda.

Comunicazioni tra client con sistema fail operational:

Nel qual caso sia stata implementata la strategia di replicazione dell'applicativo server, in caso di fallimento o crash di quest'ultimo, il client può collegarsi ad un altro server utilizzando, ad esempio, una lista comunicatagli precedentemente da esso. Questo procedimento è fail tolerant.

Nel caso in cui la strategia sopra elencata non sia implementata, allora parliamo di sistema fail operational, dato che si potrebbe sviluppare un sistema che permetta un collegamento client-to-client dopo la caduta del server a cui erano collegati. Questo introdurrebbe però alcuni problemi relativi alla consistenza delle informazioni e alla sicurezza.

3. Specifica Requisiti

3.1 Requisiti interfaccia esterna

- Interfaccia utente

L'interfaccia utente sarà costituita da una suite grafica in grado di pilotare l'utente nelle sue scelte in modo automatico.
Ogni operazione sarà richiamabile tramite la pressione di bottoni, e i risultati saranno sempre riscontrabili a video, ad esempio il caricamento con successo di un file nella lista del server è seguito dalla comparsa del nome dello stesso nel menu dei file condivisi. Gli eventuali output di debug saranno disponibili nei file definiti dall'utente nel file di configurazione.

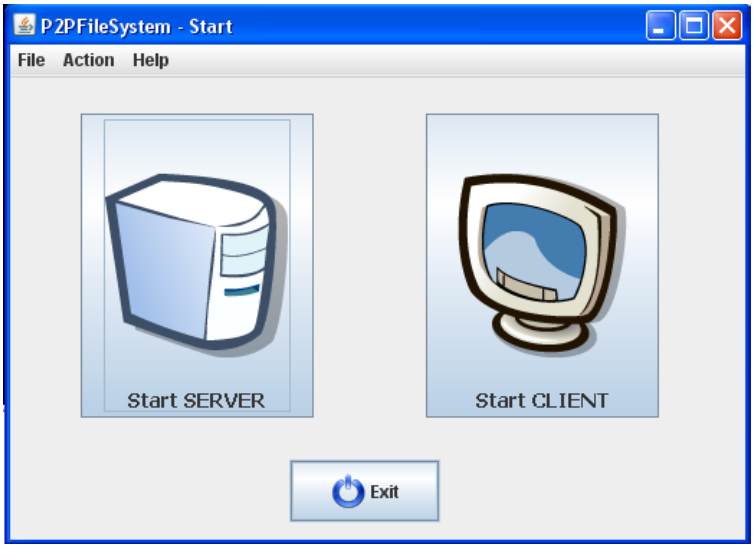
L'interfaccia utente è suddivisa in tre parti:

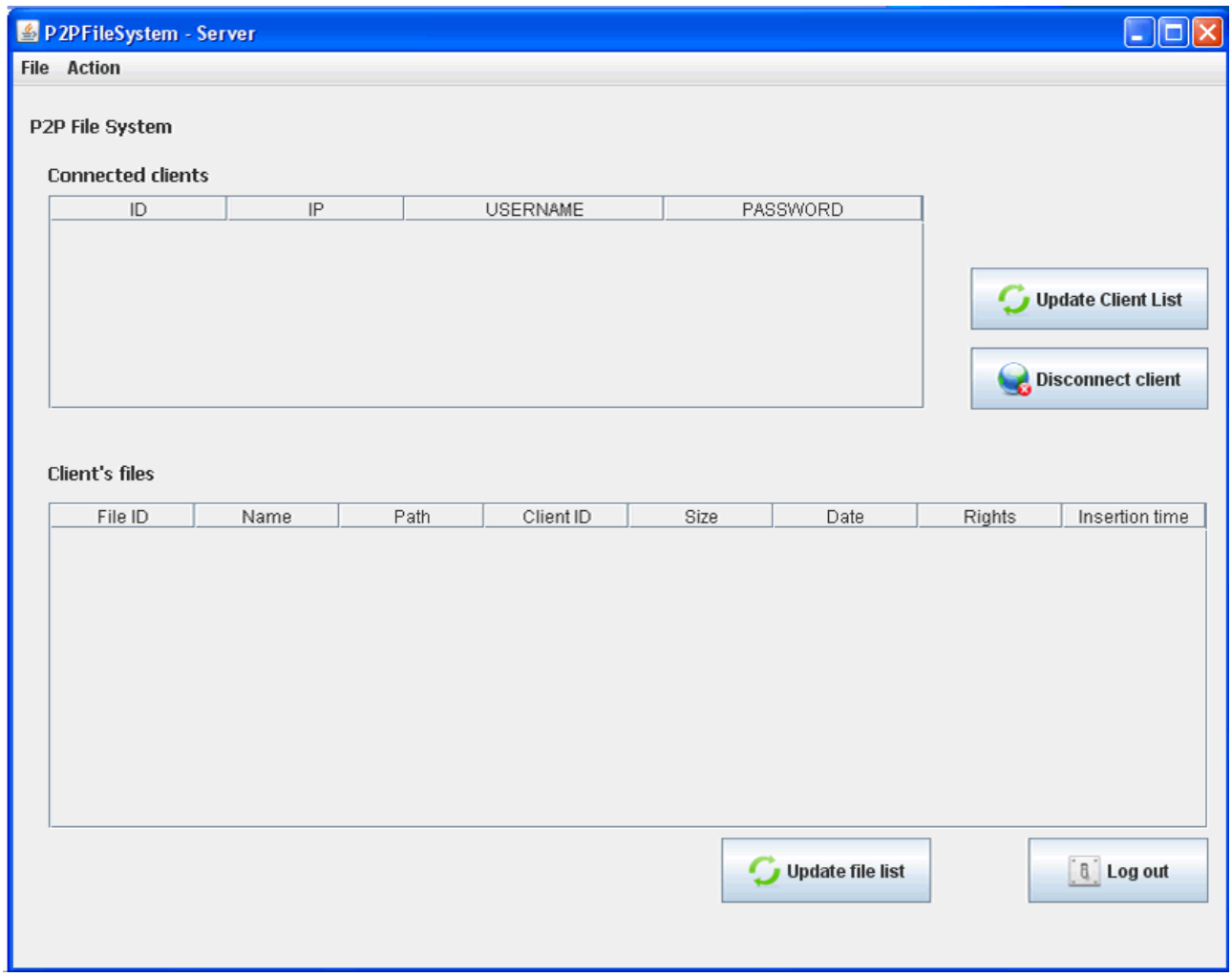
- quella in basso a sinistra permette di effettuare la ricerca in broadcast degli indirizzi dei server attivi, di inserire manualmente l'ip di un server, di effettuare le operazioni di connessione e disconnessione;

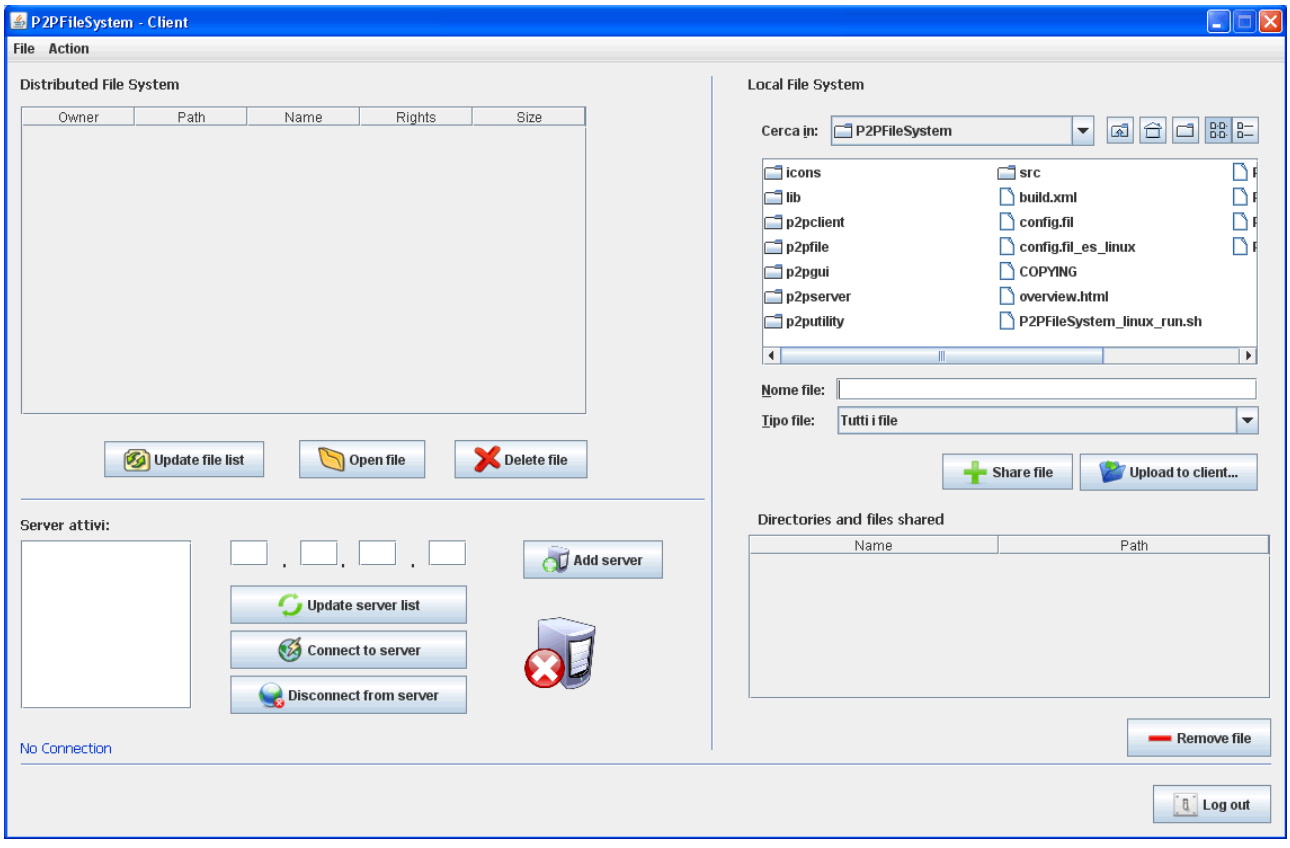
- quella in alto a sinistra consente, tramite il bottone "Update", di richiedere e visualizzare l'elenco dei file condivisi sul server; inoltre, selezionando un file dalla lista, è possibile ottenerne una copia dal bottone "Open file", o cancellarla, se i permessi lo consentono, tramite il bottone "Delete".

- quella a destra consente di selezionare i file da condividere presenti nel proprio file system locale; tramite il bottone "Upload to client..." è possibile inviare il file selezionato ad un determinato client, scelto da una lista; nella tabella in basso a destra vengono visualizzati i file che il client ha condiviso e tramite il bottone di "Remove" è possibile interromperne la condivisione.

Per maggiore chiarezza, si vedano le figure successive:







- Interfaccia hardware

L'applicazione è in grado di funzionare su praticamente tutte le piattaforme hardware moderne, a patto che possano connettersi a una qualche rete. Non esistono canoni prestativi specifici che indichino configurazioni migliori di altre.

- Interfaccia software

La piattaforma software sulla quale il programma deve andare in esecuzione non ha particolari vincoli di sistema operativo, ma richiede un'installazione di Java Development Kit 1.6 (o superiore). Il percorso per l'esecuzione del comando "rmic" (Remote Method Invocation Compiler) può essere specificato nel file di configurazione, o lasciato vuoto nel caso le variabili d'ambiente siano configurate in modo corretto.
Se l'applicazione viene lanciata lato server, è altresì necessario un supporto per MySql versione 5 (o successive) con privilegi di amministratore per creare utenti e DB (con i relativi schemi).

-Interfaccia comunicazione

La tecnologia RMI sfrutta il protocollo TCP/IP, che deve quindi essere presente nel sistema per far funzionare lo scambio di file. E' altresì necessario che la comunicazione tramite datagrammi UDP implementata a basso livello attraverso le socket, sia concessa all'interno della rete locale.

3.2 Requisiti funzionali

Attivazione di un server/client

Introduzione: al momento della partenza dell'esecuzione dell'applicazione, deve essere possibile la scelta tra l'attivazione del programma lato client/server
Input: esecuzione del programma
Processing: mandando in esecuzione l'applicazione, compare una interfaccia grafica che richiede la scelta tra l'attivazione del lato client o server. Nel primo caso comparirà un'interfaccia client dove sarà possibile connettersi a server per condividere file, nel secondo caso dovrà invece essere creata una nuova istanza di una DB dove inserire i riferimenti ai nodi connessi e ai file condivisi dai nodi, oltre che l'attivazione di una sessione RMI in ascolto su una determinata porta.
Output: attivazione dell'interfaccia grafica con le operazioni lato client/server;

Ricerca di un server su una rete locale

Introduzione: l'applicazione, in esecuzione lato client, deve poter trovare i server presenti nella rete locale ai quali collegarsi per condividere i file remoti. Tutti i server trovati dovranno essere visibili all'utente per permettergli di effettuare la scelta migliore.
Input: indirizzo del client e netmask della rete locale
Processing: cliccando sull'apposito bottone, l'applicazione cercherà i server attivi nella rete locale (la stessa sulla quale è in esecuzione) mandando una richiesta broadcast in tutti i nodi presenti nella rete sulla porta definita. Ogni risposta da parte di un server (che è in ascolto) verrà considerata un sinonimo di attività e l'indirizzo del server verrà aggiunto tra quelli a cui è possibile connettersi.
Output: lista dei server attivi a cui collegarsi

Autenticazione e accesso al FileServer

Introduzione: un utente, ottenuta una lista aggiornata dei server, deve potersi connettere a uno di questi tramite autenticazione, per poter accedere alla lista dei file condivisi attraverso ad esso
Input: indirizzo di un server, scelto da una lista in formato grafico
Processing: una volta scelto il server attraverso la GUI, in automatico l'applicazione lato client manda una richiesta di connessione RMI alla sua speculare lato server su una porta definita. Lato server, un thread sarà sempre in ascolto e provvederà a controllare le credenziali fornite dal client, che potrà autenticarsi nel sistema in modo sicuro solo se in possesso dell'hash SHA-1 della password contenuta nella lista (in formato CSV) definibile dal gestore del server.
Output: messaggio di avvenuta connessione

Gestione accesso remoto ai file:
Lato server; ricerca del file nella lista del File Server

Introduzione: una volta scelto un file, il Server dovrà rintracciarlo nella sua lista per poterne estrapolare gli attributi, tra cui l'indirizzo del nodo proprietario per poterlo contattare.
Input: nome del file richiesto
Processing: il Server riceve la richiesta fatta dal Client (attraverso l'interfaccia grafica), e cerca nella sua tabella del DB la entry corrispondente attraverso una query SQL. Gli attributi di sicuro interesse sono lo stato e l'indirizzo del nodo proprietario, oltre che il percorso del file nel filesystem del nodo.

Output: se il file è accessibile inizio della procedura di condivisione, con invio richiesta

Gestione accesso remoto ai file:
Lato server; contatto del client mittente

Introduzione: se il file risulta accessibile, il Server dovrà contattare il Client mittente per informarlo di una prossima richiesta per un determinato file
Input: richiesta di apertura di una porta in ascolto per un determinato file (con un determinato percorso)
Processing: il Server apre un thread di comunicazione via RMI con il Client Mittente, e invia la richiesta di apertura di un thread in ascolto. A questo punto il Client Mittente attiverà una nuova sessione RMI con funzionalità di sharing dove sarà lui a funzionare come Server in ascolto, in attesa della richiesta vera e propria da parte di un altro nodo. Il Server nel suo messaggio invierà anche una chiave e un valore univoci per creare una password ad hoc (utilizzando un hash SHA-1) per la connessione con il Client Ricevente.
Output: attivazione di una sessione RMI in ascolto per lo sharing, altrimenti messaggio di errore

Gestione accesso remoto ai file:
Lato server e client; condivisione della locazione del client mittente con il client ricevente

Introduzione: il Server deve dare le informazioni necessarie al Client che ha richiesto il file per rintracciare il nodo su cui il file è effettivamente memorizzato
Input: messaggio di risposta alla richiesta file, comprendente l'indirizzo del nodo da contattare
Processing: il Server risponde alla richiesta di un file remoto inviando l'indirizzo del nodo proprietario del file (estrapolandolo dal DB) e una coppia chiave-valore per creare una password univoca di accesso al Client Mittente. La ricezione di questo messaggio fa aprire al Client ricevente una nuova sessione RMI che dovrà tentare di contattare e autenticarsi col nodo remoto per la condivisione.
Output: attivazione di una sessione RMI lato client

Gestione accesso remoto ai file:
Client-to-client; connessione tra client e ricezione del file

Introduzione: una volta che il client che richiede il file viene informato della locazione del nodo proprietario della risorsa, lo deve contattare e iniziare lo scambio fisico del file
Input: richiesta del file da parte del “Client Ricevente” verso il “Client Mittente”
Processing: nel nodo che richiede il file verrà effettuata una chiamata di un metodo RMI presente sul server RMI attivo sul nodo indicato dal Server centralizzato. Se l'operazione va a buon fine, i due thread in comunicazione tra di loro possono scambiarsi il file, sempre previa autenticazione attraverso la password concordata attraverso il Server centralizzato.
Output: conferma dell'avvenuta ricezione e accesso al file in locale

Gestione accesso remoto ai file:
Client-to-client; connessione tra client e cancellazione del file

Introduzione: una volta che il client che richiede la cancellazione del file viene informato della locazione del nodo proprietario della risorsa, lo deve contattare ed effettuare la richiesta
Input: richiesta di cancellazione del file da parte del “Client Ricevente” verso il “Client Mittente”

Processing: nel nodo che richiede la cancellazione del file verrà effettuata una chiamata di un metodo RMI presente sul server RMI attivo sul nodo indicato dal Server centralizzato. Se l'operazione va a buon fine, il thread ricevente può effettuare la cancellazione del file se questo ha il permesso di scrittura e sempre previa autenticazione attraverso la password concordata attraverso il Server centralizzato.
Output: conferma dell'avvenuta cancellazione del file in locale

Gestione accesso remoto ai file:
Client-to-client; connessione tra client e invio del file

Introduzione: una volta che il client che deve inviare il file ha selezionato il client che deve riceverlo, richiede l'indirizzo della locazione del nodo ricevente al server, lo contatta e chiede di poter iniziare lo scambio fisico del file
Input: richiesta di invio del file da parte del "Client Mittente" verso il "Client Ricevente"
Processing: nel nodo che invia il file verrà effettuata una chiamata di un metodo RMI presente sul server RMI attivo sul nodo indicato dal Server centralizzato. Se l'operazione va a buon fine, i due thread in comunicazione tra di loro possono scambiarsi il file, sempre previa autenticazione attraverso la password concordata attraverso il Server centralizzato.
Output: richiesta di salvataggio del file ricevuto

Condivisione di un file:
Scelta del file da condividere

Introduzione: un Client, dopo la connessione a un server, deve poter decidere di condividere uno o più dei suoi file attraverso il server
Input: nome del file da condividere
Processing: attraverso un'interfaccia grafica, lato client viene scelto il file che si vuole condividere nella rete locale.
Output: invio della richiesta per la condivisione del file al server

Condivisione di un file:
Salvataggio del file nella lista centralizzata

Introduzione: una volta scelto un file e inviata la richiesta di condivisione, il Server dovrà salvare il percorso del file per poterlo segnalare a tutti i client della sottorete a lui sottoposta
Input: richiesta con il nome del file e il suo percorso
Processing: il Server riceve la richiesta con il percorso completo del file e tutti i suoi attributi. Attraverso una query SQL di inserimento di una entry, aggiorna la tabella da lui in possesso nel DB (se il riferimento al file non è già presente), completandola di tutte le informazioni necessarie a rintracciare il file in caso di richieste successive. Il Server informa poi il Client dell'avvenuta condivisione logica e permette di leggere la sua lista di file condivisi aggiornata.
Output: messaggio di conferma di condivisione, altrimenti messaggio di errore

Disconnessione da un server

Introduzione: un client deve potersi disconnettere da un server.
Input: richiesta di disconnessione
Processing: il Server riceve la richiesta di disconnessione e processa la cancellazione di tutte le

entry di file presenti nel suo DB e corrispondenti al nodo uscente. Una volta fatto questo, elimina il nodo dalla lista di nodi connessi e ritorna una risposta affermativa
Output: risposta affermativa sull'avvenuta disconnessione

3.3 Requisiti non funzionali

Il sistema deve essere utilizzabile da qualsiasi macchina

La potenziale utilità dell'applicazione risulta nel poter essere sfruttata in ambienti locali di lavoro in cui sia necessario condividere file. Non sempre è possibile prevedere reti locali comprendenti macchine con lo stesso sistema operativo. E' quindi consigliabile (anche se non strettamente necessario) fare in modo che l'applicazione sia portabile sulla maggior parte delle architetture.

Il sistema deve poter essere utilizzato anche da utenti con poche conoscenze informatiche

In ogni ambiente lavorativo o accademico il livello degli utenti può essere vario. Il corretto utilizzo di un software può essere determinato anche dalla capacità si essere user-friendly anche con utenti con minime conoscenze informatiche. L'utilizzo di un'interfaccia grafica esaustiva migliora la qualità del tempo di utilizzo.

Il sistema deve prevedere una politica di sicurezza

Qualsiasi software distribuito deve necessariamente dare un set minimo di garanzie di sicurezza. I canoni da rispettare possono essere più o meno restrittivi, a seconda della tipologia dell'applicazione e del contesto in cui questa deve essere eseguita. Nel nostro caso è necessario avere dei controlli di accesso ai file condivisi e ai DB, per evitare che qualche malware possa creare disturbi.

3.4 Attributi del sistema

Portabilità

Il software P2PFileSystem è progettato in linguaggio Java, sfruttando la tecnologia RMI. Java è un linguaggio che sfrutta una Virtual Machine che è installabile in tutte le architetture moderne. Rmi è una funzione di Java, e quindi portabile anch'essa. Questa combinazione risulta quindi funzionante (salvo problematiche singolari) su ogni architettura.

Affidabilità

Non essendoci nessuna forma di replicazione, la consistenza dei dati è garantita dalla loro esistenza univoca all'interno del file system di uno dei client. L'affidabilità è quindi limitata alle comunicazioni tra Client e Server del sistema. In caso di crash o chiusura di un Server, non vengono scatenati meccanismi automatici di gestione: il client si accorge della chiusura del Server appena

tenta di intraprendere una qualche operazione. In questo caso, il Client riconoscerà la chiusura, informerà l'utente attraverso l'interfaccia e svuoterà tutte le finestre grafiche associate al server.
In caso contrario, cioè di chiusura del Client, il Server viene informato attraverso una chiamata RMI della disconnessione, in modo da provvedere alla rimozione dalle liste dei file associati a quel client.

Sicurezza

I file sono accessibili in modo sicuro, in quanto è stata implementata anche la sicurezza del sistema. Per effettuare una qualsiasi operazione su file (ad esempio read, delete, etc…), e più in generale un'operazione tra client e client, il primo client deve sempre chiedere l'autorizzazione al server, il quale fornisce ad entrambi i client una stringa d'accesso con un'altissima probabilità di univocità ($1-1/2^{256}$). Nel momento in cui il secondo client riceve la richiesta, verifica se la stringa d'accesso che accompagna la richiesta è corretta, e solo in questo caso può acconsentire all'operazione. Questo garantisce che nessun utente non autorizzato dal server possa effettuare operazioni sui file e più in generale non possa richiedere operazioni ad un client.

3.5 Meccanismi Aggiuntivi

Verranno di seguito elencati alcuni meccanismi aggiuntivi (quindi non compresi nei requisiti) incorporati nell'applicativo.

Gestione permessi sui file

Al momento della condivisione, il client ha la possibilità di indicare i permessi concessi sulle modifiche ai file.
Con permesso di lettura, intendiamo la possibilità di fare una "get" del file, mentre con permesso di scrittura, forniamo ad un client il diritto di eliminare un file.
Non è risultato possibile fornire un vero e proprio diritto di write, in quanto le modifiche del contenuto e del nome del file non erano totalmente controllabili e condivisibili con il vero possessore del file. E' stato deciso di dare un'alternativa per ovviare a questa mancanza dando la possibilità, attraverso l'operazione "upload file", di mandare un'eventuale copia del file modificato direttamente al possessore della copia originale.

Politica file in condivisione

Non è stato necessario definire una vera e propria politica di accettazione per i file. Anche il problema dei file omonimi non è stato affrontato in maniera diretta, in quanto ogni file viene rappresentato dal suo nome, dal suo path nel file system del client e da un identificatore, rendendolo necessariamente univoco.

Configurazione personalizzata del sistema

E' possibile dare una configurazione personale all'intero sistema modificando i valori di default presenti nel file "config.fil". In caso contrario, il sistema utilizzerà quelli predefiniti.
Si riporta in appendice un esempio del file di configurazione.

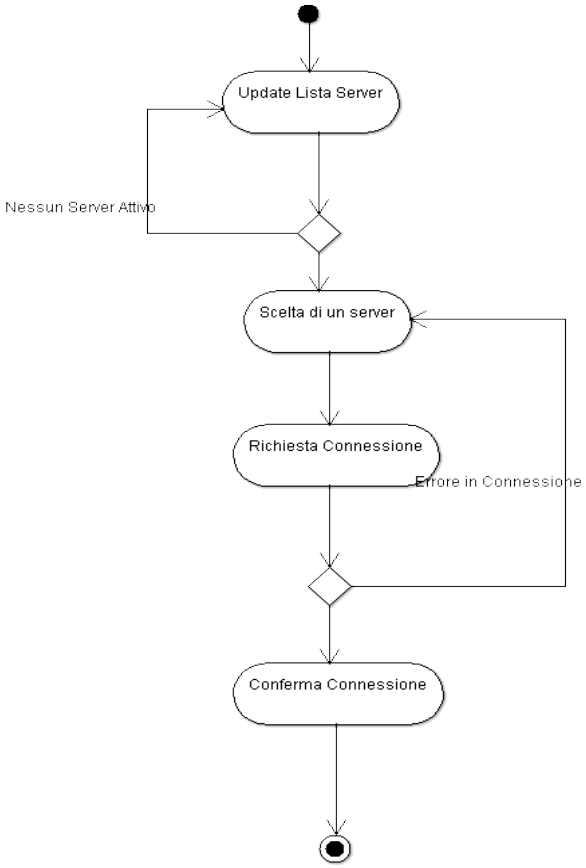
Accesso ai file condivisi

I file sono accessibili in modo sicuro, in quanto è stata implementata anche la sicurezza del sistema.

3.6 Activity Diagram

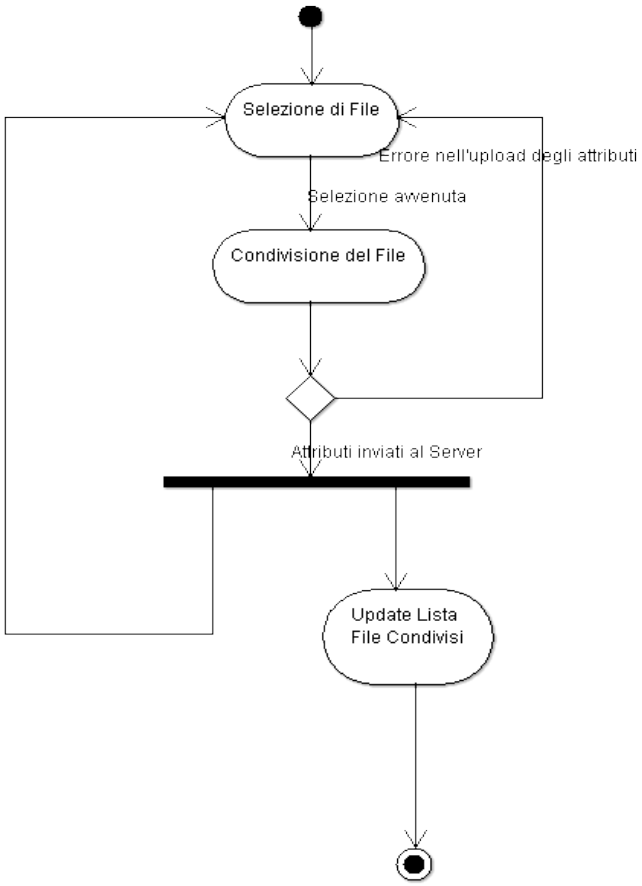
In seguito vengono proposti e commentati gli Activity Diagram di alcuni casi d'uso particolarmente significativi.

Connessione a un Server



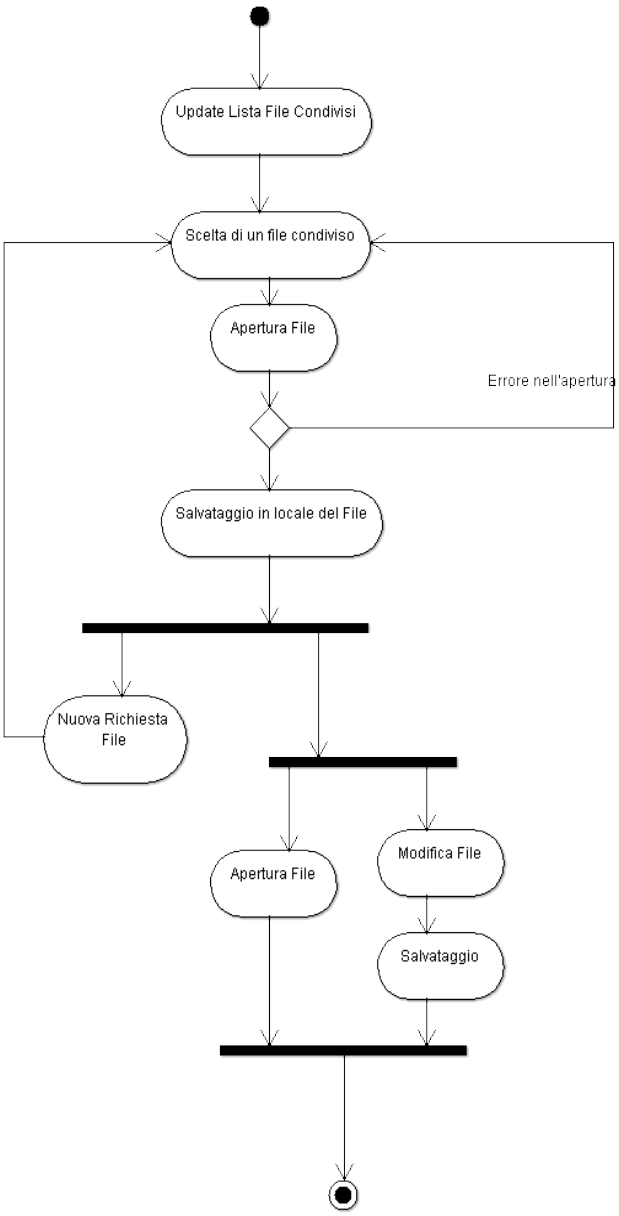
La prima operazione lato client è la ricerca di Server sulla rete locale. Questo avviene forzando l'update (manuale o automatico) della lista locale dei server. In caso non ci siano server attivi, è necessario attendere per poi ri-aggiornare la lista.
Avuta la lista, è necessario scegliere un server e richiedere la connessione. Se la richiesta va a buon fine, il server invierà un messaggio di conferma al client, altrimenti verrà notificato un errore e la procedura dovrà essere ripetuta.

Condivisione di un File



Il client per condividere un file deve prima selezionarlo. Appena effettuata la selezione, il sistema provvede a estrapolare gli attributi e a condividerli con il Server.
A questo punto, lato Client, è possibile aggiungere altri file alla lista di condivisione (ritornando quindi al punto di partenza), o richiedere al Server di inviare una lista aggiornata di tutti i file di cui possiede gli attributi.

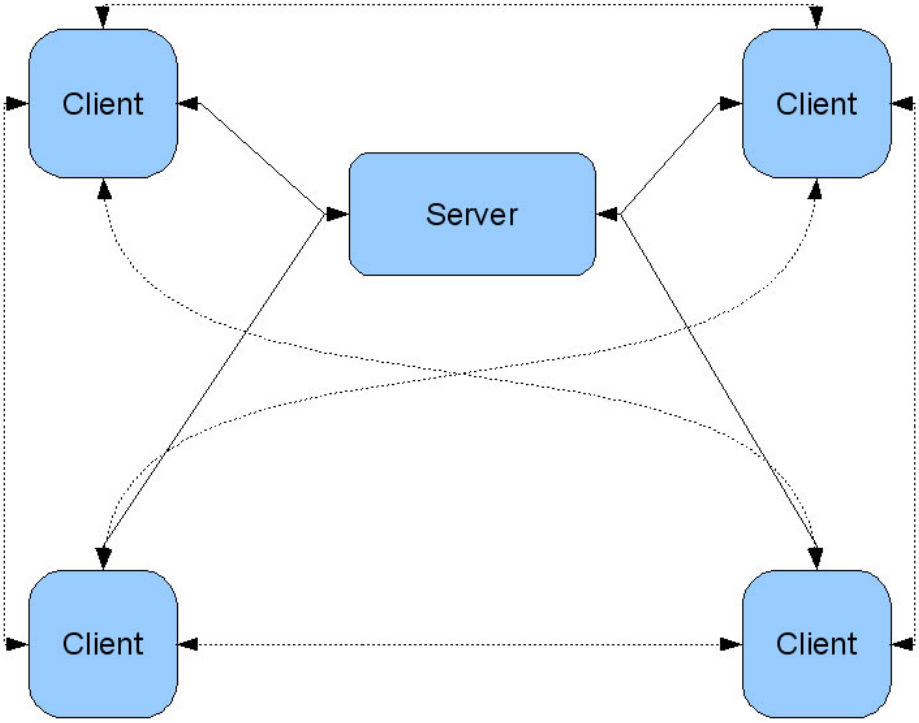
Richiesta di un file remoto



Un client può accedere a un file condiviso scegliendolo da una lista aggiornata inviatagli dal server. Una volta scelto, attraverso l'interfaccia grafica si scatenano le operazioni RMI che permettono di autenticarsi sul Client proprietario e copiare il file in locale. Se il salvataggio non va a buon fine, la procedura va ripetuta, altrimenti è possibile richiedere altri file o aprire in lettura ciò che si è scaricato.
E' altresì possibile, dopo il download del file, modificarlo e inviarne la modifica al proprietario, mantenendone la sua copia aggiornata.

A – Appendice

1. Topologia del sistema:



Come si può notare dall'immagine, il sistema presenta una topologia a stella, dove il nodo centrale (server) è collegato a tutti gli altri nodi (client). A differenza però della comunicazione server-client, la comunicazione client-client presenta una topologia totalmente connessa.

2. File di configurazione:

```
#############################################################################################
### FILE DI CONFIGURAZIONE DEL PROGRAMMA P2PFileSystem ######################################
#############################################################################################
#
# Questo file deve essere nominato 'config.fil' e deve essere leggibile.
# Ogni riga che inizia per '#' è identificata come commento, perciò ignorata dal programma.
# Se questo file viene a mancare oppure non è leggibile l'applicazione caricherà una
# configurazione standard. Se il file è corrotto potrebbe causare malfunzionamenti.
#
#############################################################################################
### INIZIO CONFIGURAZIONE: ##################################################################
#############################################################################################
#
# Percorso realtivo dei file compilati (.class)
# Default: classpath= /build/classes/
#
classpath= \build\classes\
#---------------------------------------------------------------------------
#
# Percorso assoluto dell'eseguibile rmic
# Default: #rmicpath=
#
#rmicpath= C:\Programmi\Java\jdk1.6.0_10\bin\
#
#---------------------------------------------------------------------------
#
# Percorso relativo dei file del programma
# Default: configpath= /p2pfile/
#
configpath= \p2pfile\
#
#---------------------------------------------------------------------------
#
# Nome della lista degli utenti che possono fare login sul server.
# Necessita del formato CSV. Si trova nella directory 'configpath'
# Default: userlistfile= userlist.csv
#
userlistfile= userlist.csv
#
#---------------------------------------------------------------------------
#
# Indirizzo ip di default (test su localhost)
# Default: defip= 127.0.0.1
#
defip= 127.0.0.1
#
#---------------------------------------------------------------------------
#
# Subnet Mask - Maschera della sottorete
# Default: netmask= 255.255.255.0
#
netmask= 255.255.254.0
#
#---------------------------------------------------------------------------
#
# Porta di default in ascolto del server
# Default: port= 9090
#
port= 9090
#
#---------------------------------------------------------------------------
#
# Nome dell'utente per l'accesso al DB - (la password sarà generata casulamente)
# Default: userp2p= p2puser
#
userp2p= p2puser
#
#---------------------------------------------------------------------------
#
# Nome del database necessario all'avvio e alla gestione del server
# Default: dbp2p= p2pdb
#
dbp2p= p2pdb
#
#---------------------------------------------------------------------------
#
```

```
# Numero massimo di connessioni effettuate verso questo client da altri client per inviare file
# Default: availableconn= 1
#
availableconn= 2
#
#-------------------------------------------------------------------------
#
# Tempo di attesa massimo (in millesimi di secondo - ms) per le funzioni remote
# Attenzione:  Tempi troppo brevi portano a malfunzionamenti.
#              Il valore 0 (zero) corrisponde ad attesa infinita.
#              Si consiglia un valore compreso tra 3000 e 20000 (3-20 secondi).
# Default: threadtimeout= 10000
#
threadtimeout= 10000
#
#-------------------------------------------------------------------------
#
# Numero di porta per avviare il servizio RmiRegistry
# Default: rmiregport= 1099
#
rmiregport= 1099
#
#-------------------------------------------------------------------------
#
# File dove aggiornare i messaggi di log del client (visibili nella directory 'configpath')
# Commentare la riga per disabilitare l'opzione di LOG.
# La lunghezza del nome del file deve essere superiore a 2 caratteri per essere valida.
# Default: clientlog= LOG_client.txt
#
clientlog= LOG_client.txt
#
#-------------------------------------------------------------------------
#
# File dove aggiornare i messaggi di log del server (visibili nella directory 'configpath')
# Commentare la riga per disabilitare l'opzione di LOG.
# La lunghezza del nome del file deve essere superiore a 2 caratteri per essere valida.
# Default: serverlog= LOG_server.txt
#
serverlog= LOG_server.txt
#
#-------------------------------------------------------------------------
#
# File dove aggiornare i messaggi di log generici (visibili nella directory 'configpath')
# Commentare la riga per disabilitare l'opzione di LOG.
# La lunghezza del nome del file deve essere superiore a 2 caratteri per essere valida.
# Default: genericlog= LOG_generic.txt
#
genericlog= LOG_generic.txt
#
#-------------------------------------------------------------------------
#
# Tempo di attesa massimo (in millesimi di secondo - ms) per il broadcast
# Default: bcasttimeout= 3000
#
bcasttimeout= 1000
#
#-------------------------------------------------------------------------
#
# Massima dimensione accettabile di un file in ingresso
# Esempi: 1B = 1, 1KB = 1024, 1Mb = 1048576, 1Gb = 1073741824
# Il massimo accettabile e' 2Gb = 2147483648
# Default: filemaxsizetosend= 1048576
#
filemaxsizetorecv= 1048576
#
#-------------------------------------------------------------------------
#
# Massima dimensione accettabile di un file in uscita
# Esempi: 1B = 1, 1KB = 1024, 1Mb = 1048576, 1Gb = 1073741824
# Il massimo accettabile e' 2Gb = 2147483648
# Default: filemaxsizetorecv= 1048576
#
filemaxsizetosend= 1048576
#
#-------------------------------------------------------------------------
##########################################################################################
### FINE CONFIGURAZIONE ##################################################################
##########################################################################################
```