INDICE

1. INTRODUZIONE

La condivisione di informazione e file tra utenti è una delle esigenze fondamentali che ha contribuito allo sviluppo delle reti negli anni. I file possono essere scambiati in vari modi, ad esempio come allegati in un messaggio di posta elettronica o reperiti attraverso la navigazione su Internet. *FTP* (*File transfer protocol*) è un protocollo studiato appositamente per lo scambio di file, che si basa sul modello client/server: il servizio, in esecuzione su un host, rimane in attesa di richieste di caricamento (*upload*) o scaricamento (*download*) di file.

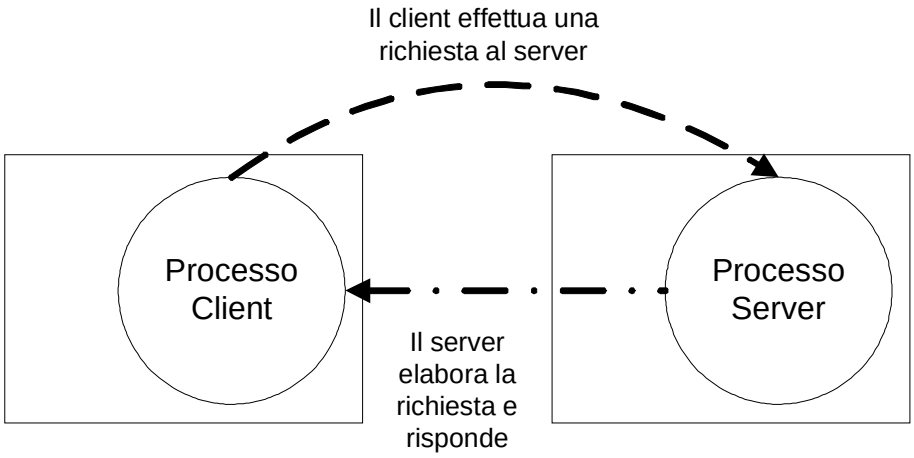

Figura 1 - Il modello client/server

Affinchè un client possa accedere al servizio remoto occorre che il server possa riconoscerlo al momento della richiesta di servizio: per questo motivo è necessaria una fase di

autenticazione del client, che tipicamente consiste nel fornire uno username e una password.

Il servizio di FTP, con altri quali le news o la posta elettronica, fa parte del livello applicazione di OSI, ed è solitamente utilizzato direttamente dagli utenti, mediante applicativi quali i client FTP o i browser.

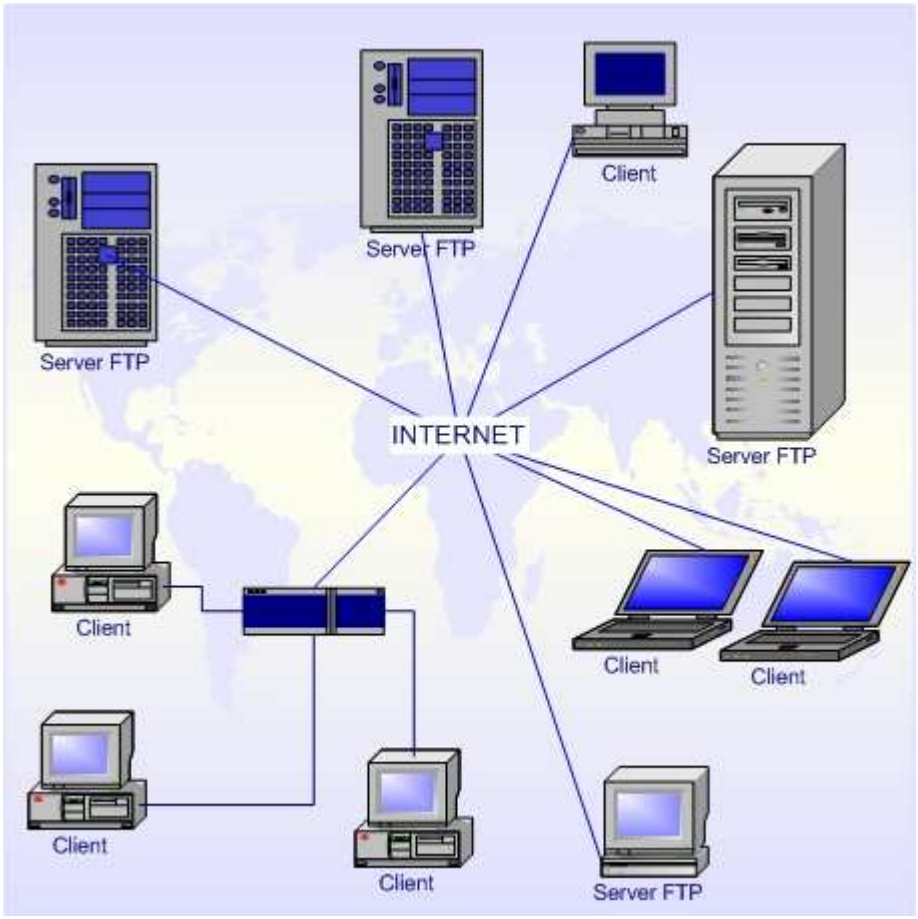

Figura 2 - Client e Server FTP

Lo scopo di questa tesi è studiare come estendere l'architettura tradizionale per introdurre un livello intermedio in cui

incapsulare intelligenza, al fine di poter supportare servizi innovativi e avanzati anche nella prospettiva di integrare più servizi in una gestione uniforme. Tutto ciò dovrebbe essere fatto, peraltro, senza modificare i protocolli esistenti. Obiettivo del lavoro è dunque realizzare questo livello intermedio per il caso del protocollo FTP, in particolare con riferimento all'upload di file. Un requisito di progetto è che questo nuovo livello si basi sulla infrastruttura di coordinazione *TuCSoN*, un'infrastruttura per la coordinazione di agenti su Internet basata su un'astrazione di coordinazione chiamata *centro di tuple*, dove per *tupla* si intende un'entità logica che contiene informazione strutturata.

Il centro di tuple è uno spazio di tuple logico, arricchito con la possibilità di specificarne il comportamento mediante un'opportuna programmazione espressa mediante particolari tuple logiche, definite dal linguaggio *ReSpecT*. Programmando opportunamente il comportamento di un centro di tuple si possono quindi realizzare le politiche di coordinazione desiderate.

Introducendo un livello intermedio tra client e server, diventa necessario disporre di entità in grado di fare da "ponte" tra l'infrastruttura di coordinazione, da un lato, e il server FTP o l'utente, dall'altro. A tale scopo TuCSoN mette a disposizione delle entità denominate *agenti* (v.di Figura 3).

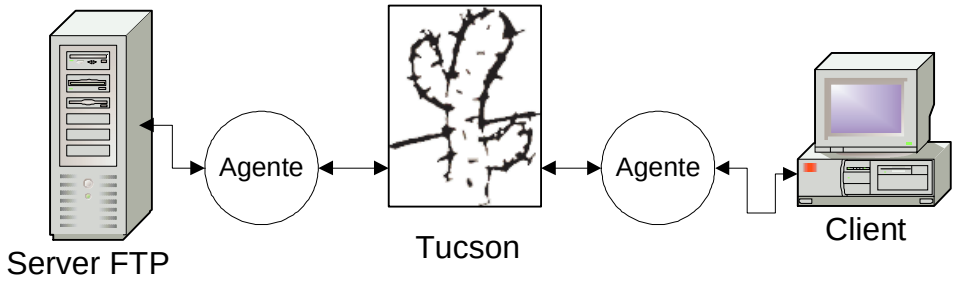

Figura 3 - Gli agenti TuCSoN

Il progetto affrontato in questa tesi prevede perciò di:

- appoggiarsi ad un qualsiasi FTP server aderente agli standard;
- sfruttare come base l'infrastruttura TuCSoN;
- realizzare un agente TuCSoN per depositare le richieste di trasferimento nei centri di tuple, sotto forma di tuple di opportuno formato;
- realizzare un ulteriore agente TuCSoN per estrarre dai centri di tuple le tuple relative alle richieste di trasferimento e, interfacciandosi con l'FTP server, provvedere all'effettivo upload dei file.

2. IL PROTOCOLLO PER IL TRASFERIMENTO DEI FILE

Data l'eterogeneità dei sistemi, nelle reti è nata da tempo l'esigenza di definire dei protocolli standard di comunicazione, ovvero delle regole standard per il trasferimento delle informazioni tra macchine. FTP è il protocollo per il trasferimento dei file e si colloca nel quarto livello del modello stack TCP, quello applicativo.

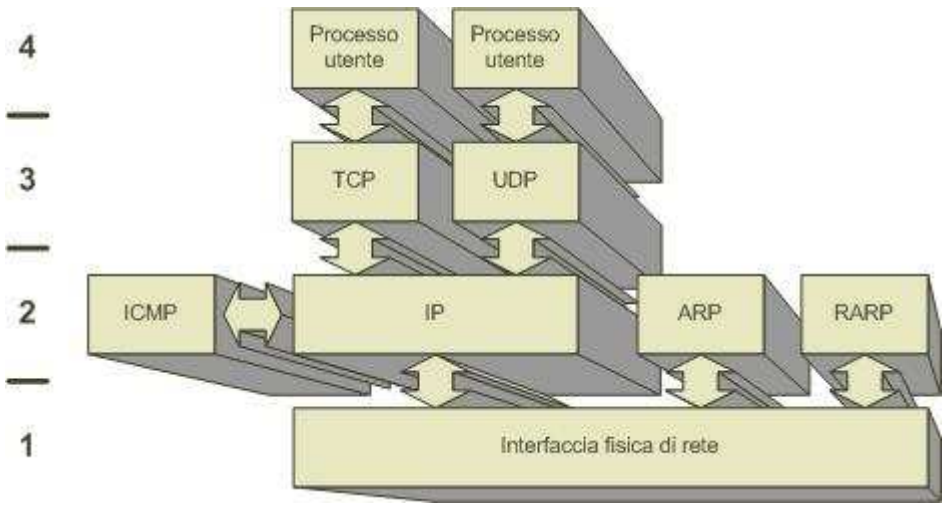

Figura 4 - Lo Stack TCP

2.1 Ftp (File transfer protocol)

2.1.1 Il modello

Il protocollo FTP è descritto completamente dal documento standard RFC 959.

Gli obiettivi di FTP sono:

- promuovere la condivisione di file;
- favorire il dialogo tra host remoti;
- tutelare un utente da variazioni dei file system sui vari host.
- trasferire i dati in maniera efficiente e affidabile.

Il protocollo FTP, sebbene utilizzabile direttamente dagli utenti attraverso un terminale, è stato progettato soprattutto per essere implementato all'interno dei programmi. La comunicazione avviene tra due host, tra i quali viene stabilita una doppia connessione TCP: una per il controllo, una per i dati. Le connessioni TCP sono, per la natura del protocollo, ordinate e affidabili. Per la connessione dati viene adottata la modalità full duplex, che supporta lo scambio di informazione nei due sensi. I comandi previsti da FTP sono di varia natura: in particolare, alcuni forniscono informazioni sull'host remoto e permettono di navigare nel suo file system remoto. Mentre la connessione di controllo è persistente, la connessione dati viene stabilita su necessità, ovvero quando un file deve essere effettivamente trasferito.

2.1.2 L'architettura

La connessione di controllo avviene tra gli interpreti comandi al client e al server. Attraverso la connessione di controllo, che segue il protocollo Telnet, vengono forniti al server i parametri per la connessione dati (ad esempio la porta), quindi il processo dati sul client può mettersi in attesa di una connessione da parte

del processo dati del server secondo i parametri specificati (v.di Figura 5).

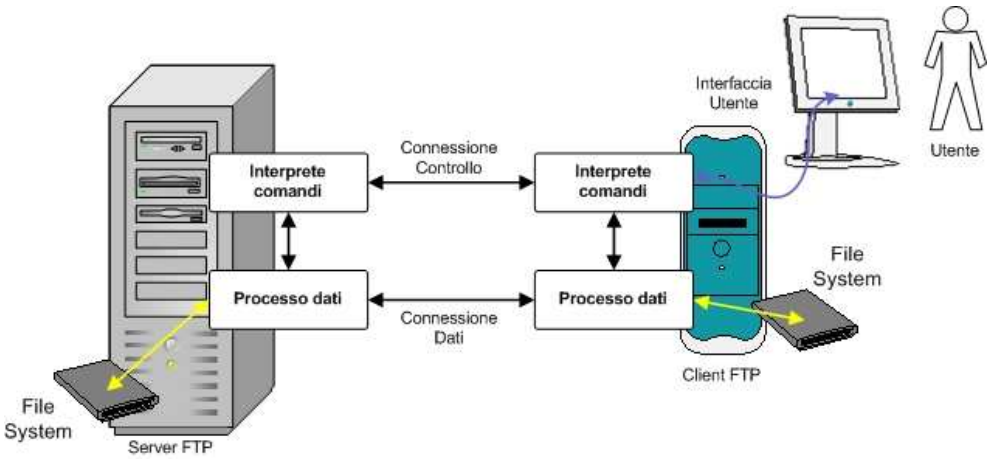

Figura 5 - L'architettura di FTP a due host

Il fatto che la connessione di controllo e la connessione dati siano distinte, suggerisce la possibilità di adottare un ulteriore schema: l'interprete comandi del client può infatti passare all'interprete comandi del server parametri relativi ad uno processo dati residente su un host distinto: in questo modo il primo host A, che ha stabilito inizialmente le connessioni di controllo, ha solo il compito di amministrare la comunicazione, mentre la connessione dati viene a crearsi tra due host B e C (v.di Figura 6).

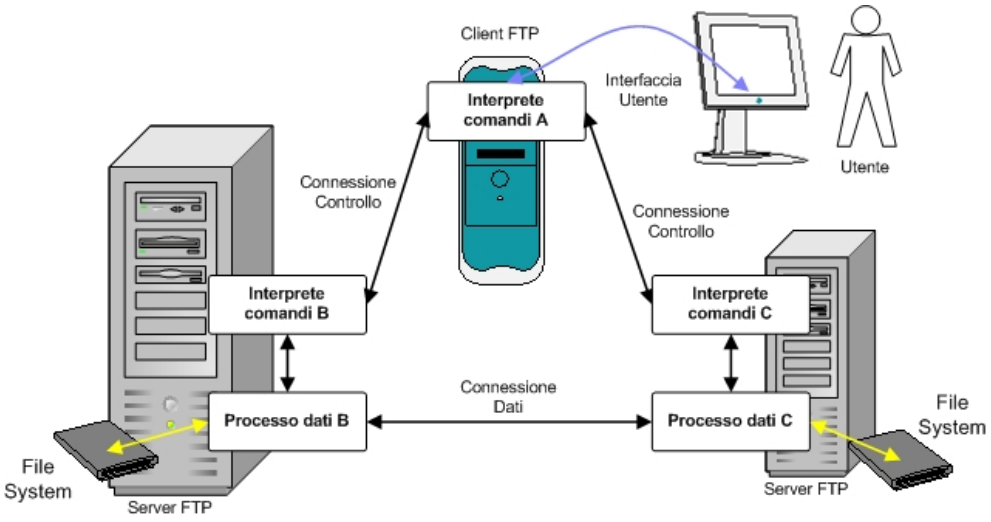

Figura 6 - L'architettura di FTP a tre host

Il protocollo richiede che le connessioni di controllo restino sempre attive per tutta la durata del trasferimento; è responsabilità dell'interprete comandi sul client chiudere le connessioni di controllo solo quando si è finito di usare il servizio: se la connessione di controllo è stata chiusa senza utilizzare i comandi opportuni il server può infatti terminare la connessione dati.

2.1.3 Funzioni per il trasferimento dei dati

2.1.3.1 Rappresentazione e memorizzazione dei dati

Nel mondo delle reti, uno dei problemi più sentiti è quello dell'eterogeneità e della diversità di rappresentazione dei dati sui vari nodi; la connessione di controllo risolve questo problema sfruttando il protocollo Telnet: i comandi e le risposte sono infatti sequenze di caratteri ASCII NVT (*Network Virtual*

Terminal). Il problema è stato risolto anche per la connessione dati, definendo alcuni tipi di dato, alcune strutture e alcune modalità di trasmissione dati FTP standard.

2.1.3.1.1 Tipi di DATO

Il tipo di dato viene dichiarato con il comando TYPE.

- ASCII: è l'ASCII NVT a otto bit ed è il predefinito; ogni implementazione FTP deve supportarlo. E' utilizzato tipicamente per il trasferimento di file di testo. In accordo con lo standard NVT la sequenza <CRLF> (*Carriage Return*, *Line Feed*) è utilizzata per indicare la fine di una riga di testo.
- EBCDIC: questo tipo, non molto diffuso, in cui i caratteri sono rappresentati a otto bit, è da utilizzare solo per trasmissioni tra host che usino localmente questo formato per la rappresentazione dei caratteri. Per l'interruzione di linea viene utilizzato il carattere <NL> (*New Line*).
- IMAGE: i dati sono inviati come sequenze contigue di bit che, per il trasferimento, vengono divisi in pacchetti di un byte. Il ricevente dovrà reinterpretarli come sequenza unica di bit. Il tipo IMAGE necessita di un *padding* (riempimento) per il file (o per ciascun record, se il file da trasferire ha una struttura a record) per arrivare ad un blocco di dati di dimensioni opportune. Il riempimento, che consiste di bit tutti a zero, può essere inserito solo alla fine del file o di ciascun record.

- LOCAL: i dati sono trasferiti in pacchetti di byte logici della dimensione definita dall'utente.

Per i tipi ASCII e EBCDIC il comando TYPE accetta anche un secondo parametro, detto FORMAT CONTROL, che indica il tipo di formato associato al file. Un carattere può essere infatti trasmesso per tre motivi essenziali: la stampa, la memorizzazione, l'elaborazione. Se ad esempio si sta inviando un file per la stampa, l'host ricevente deve sapere come formattare verticalmente il documento. Se si trasmette un file per la sola memorizzazione, esso deve poter essere recuperato esattamente uguale a come è stato inviato. Per poter soddisfare queste esigenze, i due tipi "testuali" vengono completati con il parametro relativo al formato, che può essere:

- NON PRINT: è il predefinito, e indica che il file non contiene informazioni relative al formato. Si utilizza principalmente quando l'unico scopo del trasferimento è la memorizzazione o l'invio di informazione ad applicazioni remote.
- TELNET FORMAT CONTROLS: il file contiene informazioni di controllo ASCII o EBCDIC, quali <CR>,<LF>,<NL>. In alternativa a <NL> si può utilizzare la sequenza <CRLF>.
- CARRIAGE CONTROL (ASA). Il file contiene caratteri di controllo di formato descritti nello standard ASA.

2.1.3.1.2 Strutture dati

In aggiunta ai diversi tipi di rappresentazione dei dati, FTP permette di specificare anche la struttura del file con il comando STRU. Le strutture definite in FTP sono tre: la **struttura a file**, in cui il file è considerato una sequenza contigua di byte di dati, la **struttura a record**, in cui il file è composto da una sequenza di record, e la **struttura a pagina**, in cui il file non è altro che un insieme di pagine indipendenti. La struttura a file è quella predefinita, mentre la struttura a record è accettata solo per i file di testo. La struttura a pagina è sicuramente la più complessa e viene utilizzata soprattutto per i file ad accesso casuale; alle pagine sono associati degli *header* che contengono vari parametri, quali la lunghezza dell'*header*, l'indice della pagina, il numero di byte logici presenti nella pagina, e il tipo di pagina.

2.1.3.2 Preparazione della connessione

Il primo passo per trasferire dati consiste nello stabilire la connessione dati sulle porte appropriate e scegliere i parametri per il trasferimento. I processi sul client e sul server hanno entrambi una porta dati predefinita, che per il primo è la stessa porta utilizzata per il controllo, mentre per il secondo è quella precedente a quella utilizzata per la connessione di controllo. Per specificare una porta diversa da quella di default, l'interprete comandi del client deve utilizzare il comando PORT. Per il trasferimento vengono utilizzati pacchetti di un byte. Il processo

dati attivo sul client rimane passivamente in attesa di connessione sulla porta dati indicata al server.

Quando la connessione dati avviene tra due server, allora l'interprete comandi del client deve "ordinare" ad uno dei due di attendere la connessione che il secondo stabilirà.

Quando il processo dati sul server stabilisce la connessione ha inizio il trasferimento, e l'interprete comandi del server invia una conferma a quello del client. In generale è responsabilità del server mantenere e amministrare la connessione dati. Una eccezione a questo si ha quando il processo dati del server sta inviando i dati in una modalità di trasferimento che richiede la chiusura della connessione per indicare la file del file. Altri casi in cui il server ha l'obbligo di terminare la connessione sono:

- quando riceve un comando di ABORT da parte dell' interprete comandi del client;
- quando l'interprete comandi del client invia un comando di cambio di porta;
- quando l'interprete comandi del client termina la connessione di controllo;
- quando si verifica una condizione di errore critico.

2.1.3.3 Amministrazione della connessione

Tutte le implementazioni di FTP devono supportare l'utilizzo della porta di connessione dati predefinita, e solo l'interprete comandi del client può decidere di utilizzare porte diverse. Per specificare una porta non standard lato client, l'interprete

comandi del client deve utilizzare il comando PORT; mediante il comando PASV può anche richiedere al server di identificare una porta non di default per richieste da parte di un altro server.
Talvolta, quando si utilizza una modalità *stream*, la fine del file è indicata dalle chiusura della connessione. Questo causa alcuni problemi se all'interno della stessa sessione FTP si vogliono trasferire più file, in quanto il protocollo TCP, per garantire comunicazioni affidabili, prevede di tenere occupate le porte per un certo tempo di *hold* dopo la chiusura della connessione. Pertanto la connessione sulle stesse porte non può essere ripristinata nell'istante successivo alla fine del trasferimento di un file. Le soluzioni a questo problema sono due: negoziare un'altra porta, oppure utilizzare una differente modalità di trasferimento.

2.1.3.4 Modalità di connessione

Il protocollo FTP prevede tre modalità di connessione: una che formatta i dati e permette procedure di ripristino, una che comprime i dati per trasferimenti più efficienti, e infine una che trasferisce i dati senza elaborarli quasi per niente. Tutti i trasferimenti devono essere completati con un EOF (*End Of File*), che può essere immesso esplicitamente oppure essere rappresentato dalla chiusura della connessione. Per i file con struttura a record vengono resi espliciti tutti gli EOR (*End Of Record*), incluso quello finale. Per i file trasferiti con struttura a

pagine viene trasmessa una pagina di tipo *last*. Entrando più nello specifico:

- nella modalità STREAM, i dati sono trasmessi sotto forma di stream di byte; non c'è restrizione sulla rappresentazione usata; se la struttura è a record EOR ed EOF sono rappresentati ciascuno da due byte di controllo. Se la struttura è di tipo FILE, l'EOF è indicato dalla chiusura della connessione da parte dell'host che invia i dati; in questa configurazione la modalità STREAM è inaffidabile, in quanto non si sarà mai sicuri di avere ricevuto correttamente il file;
- nella modalità BLOCK, il file è trasmesso come una serie di blocchi di dati preceduti da uno o più byte di intestazione, che contiene un campo *count* e un campo *descriptor code*. Il count indica la lunghezza totale del blocco di dati in byte, mentre il descriptor code contiene informazioni sull'ultimo blocco nel file o nel record, informazioni per il ripristino in seguito ad errori, informazioni sull'affidabilità dei dati.
- la modalità COMPRESSED è un metodo utile per ottenere migliori prestazioni di banda nelle trasmissioni su vasta scala, ad un piccolo costo extra di elaborazione.

Per i dettagli sulle tecniche di trasmissione si rimanda all'RFC 959.

2.1.3.5 Procedure di recupero e ripristino in caso di errore

Il problema del controllo degli errori non è gestito direttamente dal protocollo FTP; questo compito viene infatti demandato al protocollo sottostante, TCP. Per le modalità BLOCK e COMPRESSED è comunque prevista una procedura di recupero, per tutelare gli utenti da gravi malfunzionamenti del sistema. Essa richiede che il server inserisca nel flusso dei dati dei marcatori, che possono essere estratti dal ricevente che, in caso di fallimento, possa ripartire richiedendo i dati a partire dall'ultimo marcatore valido.

2.1.4 Funzioni per il trasferimento dei file

2.1.4.1.1 Comandi FTP

Segue un breve elenco dei comandi FTP; tutti gli argomenti ad essi passati sono stringhe TELNET.

2.1.4.1.2 Comandi per il controllo dell'accesso

- USER (USER NAME): specifica il nome dell'utente.
- PASS (PASSWORD): segue USER e specifica la password di utente.
- ACCT (ACCOUNT): è un comando di autenticazione che può essere usato sia da solo sia con USER.
- CWD (CHANGE WORKING DIRECTORY): cambia la directory di lavoro corrente.

- CDUP (CHANGE TO PARENT DIRECTORY): è un caso particolare di CWD, ed è utilizzato per passare alla directory superiore rispetto a quella corrente.
- SMNT (STRUCTURE MOUNT): è utilizzato per effettuare dei *mount* senza alterare le informazioni di login o di account.
- REIN (REINITIALIZE): ripristina lo stato iniziale della comunicazione resettando tutti i parametri e le informazioni di login; riporta la comunicazione allo stato in cui si trova appena è stata stabilita la connessione.
- QUIT (LOGOUT): se non è in atto alcun trasferimento chiude la connessione, altrimenti essa rimarrà aperta in attesa di una risposta e verrà poi terminata dal server.

2.1.4.1.3 Comandi per i parametri di trasferimento

I comandi per i parametri di trasferimento sono PORT (DATA PORT), PASV (PASSIVE), TYPE (REPRESENTATION TYPE), STRU (FILE STRUCTURE) e MODE, e sono già stati descritti precedentemente.

2.1.4.1.4 Comandi per il servizio FTP

- RETR (RETRIEVE): è il commando per recuperare un file dal server.
- STOR (STORE): commando per memorizzare un file sul server. Se il file esiste viene sostituito dal nuovo file.

- STOU (STORE UNIQUE): è analogo a STOR, con la differenza che il file viene creato con un nome unico in quella directory.
- APPE (APPEND): esegue un'operazione di *append* di dati su un file specificato; se il file non esiste viene creato.
- ALLO (ALLOCATE): questo commando può essere richiesto da alcuni server per allocare sufficiente memoria di massa per memorizzare il file.
- REST (RESTART): comando utilizzato per far ripartire il flusso dei dati dall'ultimo marcatore valido.
- RNFR (RENAME FROM): è utilizzato per dichiarare il vecchio path del file che si sta per rinominare; deve essere seguito da RNTO.
- RNTO (RENAME TO): è utilizzato per dichiarare il nuovo path per il file specificato con RNFR che lo precede.
- ABOR (ABORT): serve per richiedere al server di terminare l'esecuzione dell'ultimo comando.
- DELE (DELETE): serve per cancellare il file specificato.
- RMD (REMOVE DIRECTORY): cancella una directory.
- MKD (MAKE DIRECTORY): crea una directory.
- PWD (PRINT WORKING DIRECTORY): nella risposta a questo comando viene inserito il nome della directory di lavoro corrente.
- LIST (LIST): se il path contiene una directory o un gruppo di file il server risponde con una lista di file. Se viene

specificato un file il server risponde con le informazioni relative al file. Se non è specificato un argomento il server risponde con il contenuto della directory corrente o predefinita.

- NLST (NAME LIST): il comando è simile al precedente, ma ritorna solamente una lista di nomi.
- SITE (SITE PARAMETERS): è utilizzato per usufruire di alcuni servizi specifici del sistema del server, non abbastanza universali da essere inclusi nel protocollo.
- SYST (SYSTEM): è utilizzato per recuperare informazioni sul tipo di sistema operativo presente sul server.
- STAT (STATUS): causa l'invio di una risposta di stato sulla connessione di controllo, sotto forma di risposta.
- HELP (HELP): causa l'invio lungo la connessione di controllo da parte del server di utili informazioni riguardo alla sua implementazione.
- NOOP (NOOP): richiede l'invio di un OK da parte del server.

2.1.4.2 Risposte FTP

Le risposte FTP sono stringhe Telnet formate da un codice di tre cifre (xyz), seguito da uno spazio e da una linea di testo, e terminate con il codice di *end of line*. Ogni cifra ha un significato ben specifico: la prima denota se la risposta è positiva, negativa, o incompleta; la seconda denota il tipo di risposta, e la terza aggiunge informazioni ulteriori relativamente all'azione eseguita.

La prima cifra può assumere i seguenti valori (con y e z qualsiasi):

- 1yz: è la conferma positiva intermedia, in cui si avvisa il client che la richiesta è stata servita, ma è necessario attendere una risposta ulteriore per proseguire.
- 2yz: è la conferma positiva definitiva, che indica che l'azione è stata eseguita correttamente.
- 3yz: indica che il comando è stato accettato, ma il server è in attesa di ulteriori informazioni.
- 4yz: indica che il comando non è stato accettato, ma la condizione di errore è temporanea e il client può richiedere nuovamente l'azione.
- 5yz: il comando non è stato accettato, poichè l'azione richiesta non può essere eseguita.

La seconda cifra può assumere i seguenti valori (con x e z qualsiasi):

- x0z: queste risposte si riferiscono ad errori di sintassi, comandi sintatticamente corretti che non si adattano ad alcuna categoria funzionale, comandi superflui o non implementati.
- x1z: sono risposte a richieste di informazione.
- x2z: sono le riposte che si riferiscono alle connessioni, controllo o dati.

- x3z: risposte per il login e lo procedure di account.
- x4z: ancora non specificato.
- x5z: queste risposte indicano lo stato del file system del server.

La terza cifra, come accennato, aggiunge ulteriori informazioni relativamente all'azione eseguita. Alcuni esempi di codici tipici sono:

- 200: comando di ok.
- 500: errore di sintassi.
- 220: servizio in attesa di un nuovo utente.
- 221: chiusura della connessione di controllo da parte del server.
- 225: connessione dati stabilita: nessun trasferimento in atto.
- 226: chiusura della connessione dati con azione del file eseguita con successo.
- 227: passaggio alla modalità *passive*.
- 230: utente autenticato.
- 331: user name ok, necessaria password.

3. L'INFRASTRUTTURA DI COORDINAZIONE TUCSON

TuCSoN è un'infrastruttura per la coordinazione di agenti su Internet basata su un'astrazione di coordinazione chiamata *centro di tuple*. Una *tupla logica* è una struttura dati costituita da un nome e da una lista di argomenti. Il centro di tuple è uno spazio di tuple arricchito con una specifica di comportamento; programmando tale comportamento, mediante il linguaggio di specifica ReSpecT, si possono realizzare le politiche desiderate. Tale specifica assume la forma di reazione agli eventi di comunicazione. Il linguaggio di coordinazione di TuCSoN fornisce agenti con una doppia percezione dello spazio di interazione di TuCSoN:

- come uno spazio globale fatto di mezzi di coordinazione (centri di tuple);
- come una collezione di spazi locali associati ai nodi Internet.

Questo si adatta ad entrambi i ruoli degli agenti Internet, vale a dire, come entità network-aware o entità network-unaware.

E' possibile interagire con l'infrastruttura inserendo e recuperando le tuple dai centri di tuple di un certo nodo e cambiare il loro comportamento. E' possibile fare ciò attraverso un programma Java o Prolog oppure anche tramite un'interfaccia grafica (CLIAgent, v.di Figura 7) che permette di invocare

direttamente le primitive del linguaggio di coordinazione su un dato centro di tuple residente su un dato nodo della rete.

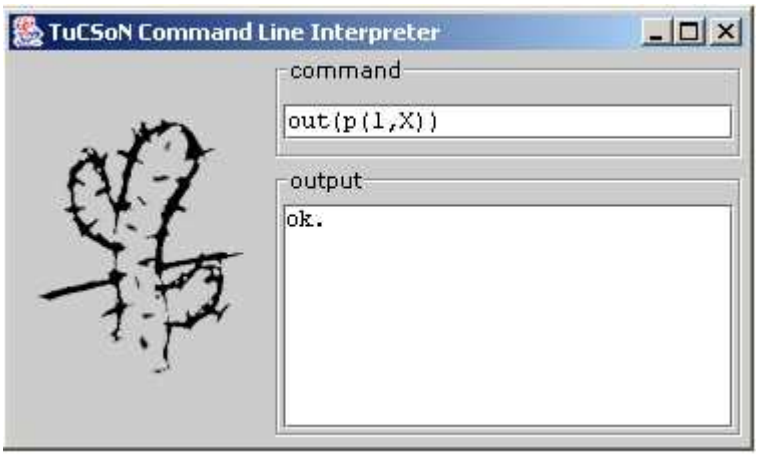

Figura 7 - CLIAgent

Le primitive si distinguono in:

- primitive per la manipolazione delle tuple (**out**, **in**, **rd**, **inp**, **rdp**);
- primitive per cambiare il comportamento del centro di tuple (**getspec** e **setspec**).

Più precisamente le primitive di manipolazione delle tuple sono:

- **out**: inserisce la tupla specificata nel centro di tuple;
- **in**: legge e rimuove una tupla che corrisponde alla tupla specificata (ha un comportamento bloccante);
- **rd**: legge una tupla che corrisponde alla tupla specificata (ha un comportamento bloccante);
- **inp**: legge e rimuove una tupla che corrisponde alla tupla specificata (ha un comportamento non bloccante);

- **rdp**: legge una tupla che corrisponde alla tupla specificata (ha un comportamento non bloccante).

Esempi:

out(p(1,hello))
Inserisce la tupla p(1,hello) nel centro di tuple.

in(p(_,hello))
Legge e rimuove la tupla il cui nome è p, ignorando il primo argomento e il cui secondo argomento è hello.

rd(p(X,Y))
Legge una tupla che ha due argomenti; X e Y sono variabili a cui saranno associati i valori degli argomenti della tupla con cui è stata verificata la corrispondenza.

set spec('reaction(out(X),(out_r(backup(X))))')
Questa primitiva programma il centro di tuple in modo che inserisca una tupla di backup ogni volta che una qualsiasi tupla viene inserita nel centro di tuple. E' possibile verificare ciò eseguendo una operazione out, ad esempio:

out(array(index(1),value(tomato)))

e quindi provando a rileggere la tupla di backup che dovrebbe essere stata generata:

rd(backup(X))

Se tutto è ok, la risposta sarà:

answer: backup(array(index(1),value(tomato)))

Se le primitive devono essere applicate alle tuple di un centro di tuple che non è quello di *default* residente sulla macchina su cui si sta lavorando, bisogna specificare, prima della primitiva, il nome del centro di tuple ed eventualmente l'indirizzo dell'host su cui esso risiede, secondo questa sintassi:

TCName @ TCAddress ? **op** (Tuple)

TCName rappresenta il nome del centro di tuple e *TCAddress* l'indirizzo o il nome dell'host su cui risiede il nodo TuCSoN.

3.1 Accesso a TuCSoN

TuCSoN mette a disposizione una serie di primitive che permettono di usufruire del servizio da applicazioni scritte in Java e in tuProlog, l'ambiente Prolog fornito con l'infrastruttura. Per accedere all'infrastruttura TuCSoN, deve essere acquisito e usato un contesto di coordinazione di agente (ACC), che rappresenta un sorta di interfaccia per alcuni contesti di coordinazione nello spazio di coordinazione supportato dall'infrastruttura. Essenzialmente, un ACC definisce quali azioni sono permesse (o meglio, quali azioni sono proibite) e può essere diverso secondo le proprietà/preferenze degli agenti (ruolo,

contesto, profilo, preferenze) e secondo le dinamiche di organizzazione/ambiente.

3.1.1 Tucson e Java

Di seguito si elencano alcuni modi per acquisire un ACC in Java.

a) Si acquisisce un contesto di coordinazione di default per un agente anonimo:

```
TucsonContext cn= Tucson.enterDefaultContext();
```

b) Si acquisisce un contesto di coordinazione di default per l'agente denominato "Goofy" :

```
AgentId agent=new AgentId("Goofy");
TucsonContext cn= Tucson.enterContext(new
              DefaultContextDescription(agent));
```

4. L'ARCHITETTURA DEL SISTEMA

4.1 L'architettura classica di FTP

L'architettura classica di un sistema FTP è di tipo *client/server*. Questo modello è così denominato a causa del diverso ruolo che viene svolto dai due partecipanti alla connessione di rete: il server è l'entità in grado di fornire un certo servizio, il client l'entità che effettua le richieste di servizio. A livello implementativo, il server è essenzialmente un processo legato ad una porta, che rimane sempre in attesa di nuove richieste, in seguito alle quali svolgerà una data attività o fornirà un certo risultato. Il client è un processo che, legatosi ad una certa porta, si connette al server per usufruire del servizio specifico. Quindi il server è una entità che ha esistenza autonoma in un certo luogo e non ha idea di quali saranno i potenziali clienti; viceversa il cliente deve conoscere perfettamente dove reperire il servizio, perciò i nomi (o gli indirizzi) e le porte dei server devono essere note a priori ai client. Lo standard stabilisce che il servizio di FTP, e in particolare la connessione di controllo, venga fornito sulla porta 21, mentre la connessione dati è solitamente (ma non obbligatoriamente) stabilita sulla porta 20.

Dal lato del client, il modello prevede uno schema a tre livelli: il primo è l'interfaccia utente, che fornisce un'astrazione del servizio facilmente utilizzabile da un generico utente senza dover conoscere il protocollo. L'interfaccia pilota l'interprete comandi, che governa il processo dati e si interfaccia con l'interprete

comandi lato server mediante la connessione di controllo. Infine, il processo dati si incarica del trasferimento dati vero e proprio; attende una connessione da parte del server, si interfaccia con il processo dati del server lungo la connessione dati e con il file system locale al cient. La seguente figura mostra l'architettura di FTP a due host.

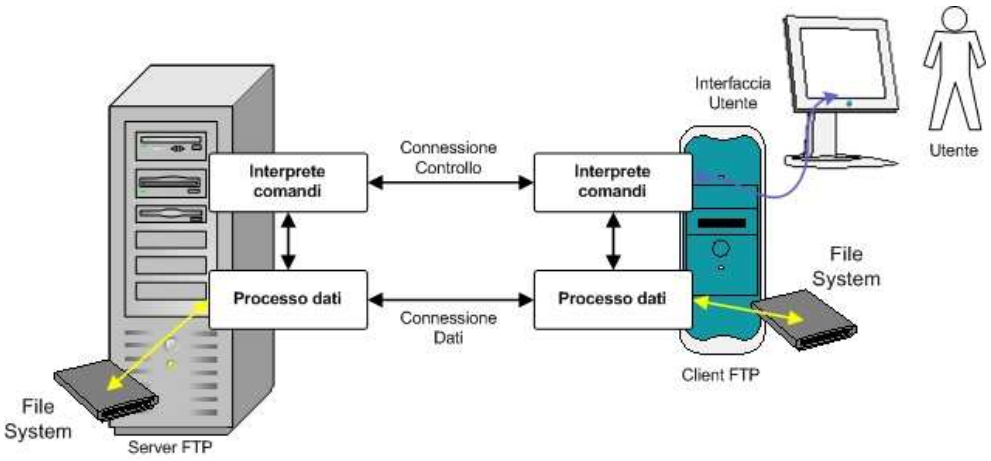

Figura 8 – L'architettura di FTP a due host

Come accennato precedentemente il processo dati potrebbe trovarsi anche su un host remoto al client (v.di Figura 9).

Dal lato del server il modello prevede due livelli: l'interprete comandi e il processo dati; il primo, analogamente all'interprete comandi lato client, si interfaccia alla connessione di controllo e comanda il processo dati; il secondo provvede a stabilire la connessione dati e si interfaccia con il file system locale al server.

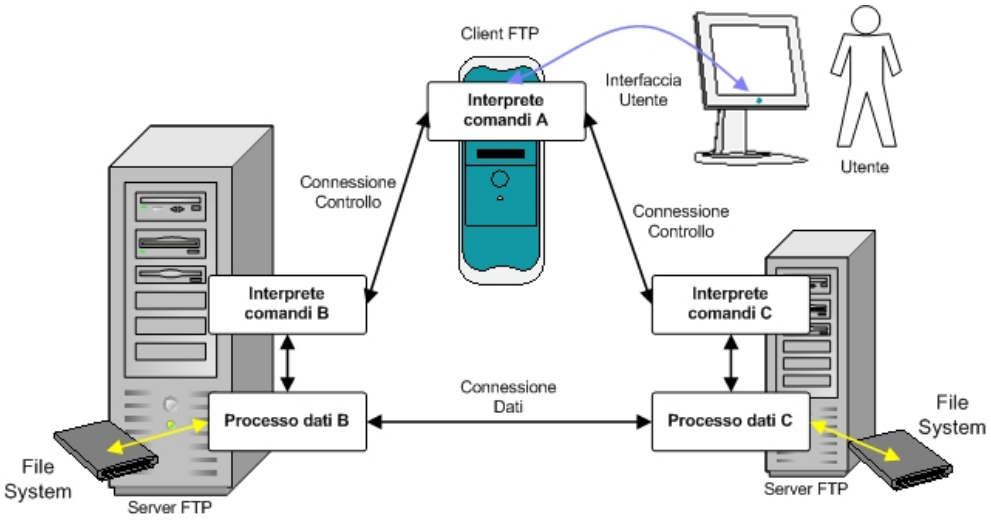

Figura 9 - L'architettura di FTP a tre host

4.2 Il nuovo approccio

Per supportare nuove caratteristiche quali intelligenza e servizi personalizzati, con applicazione di regole e politiche, il modello di architettura tradizionale non è più sufficiente, perchè lo scopo del progetto è riuscire ad introdurre queste nuove caratteristiche senza modificare i server e i protocolli esistenti. Per questo motivo è necessario estendere la tradizionale arcfhitettura FTP, ed introdurre un livello intermedio tra il client e il server. In particolare, in questa tesi si studia la possibilità supportare questo nuovo livello utilizzando l'infrastruttura di coordinazione TuCSoN che permette, programmando opportunamente il suo comportamento, di realizzare le politiche desiderate.

Il progetto affrontato in questa tesi prevede perciò di:

- appoggiarsi ad un qualsiasi FTP server aderente agli standard;
- appoggiarsi sull'infrastruttura TuCSoN;
- realizzare un agente TuCSoN per depositare le richieste di trasferimento nei centri di tuple, sotto forma di tuple di opportuno formato;

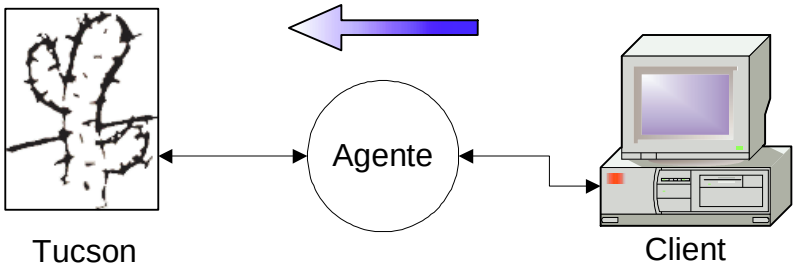

Figura 10 - Fase di trasferimento delle richieste

- realizzare un ulteriore agente TuCSoN, che estragga dai centri di tuple le tuple relative alle richieste di trasferimento e, interfacciandosi con l'FTP server, provveda all'effettivo upload dei file.

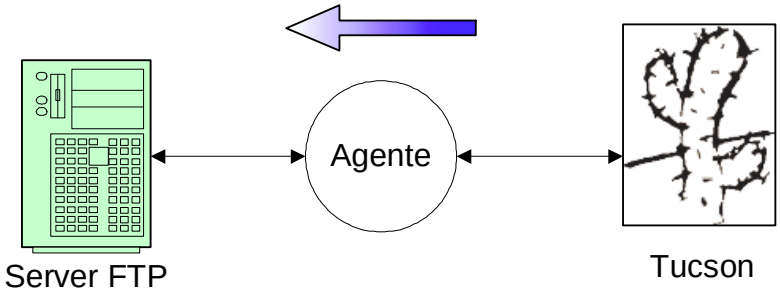

Figura 11 - Fase di upload

La opportunità fornita da TuCSoN di utilizzare un numero illimitato di centri di tuple, con un numero illimitato di tuple al loro interno, anche con schemi differenti, apre un ampio ventaglio di possibilità implementative.

Prima di passare ad esaminare le più significative, è necessario inquadrare il livello al quale si collocano gli agenti in esame, ad esempio stabilire per il sistema il significato del termine “utente”: l’utente potrebbe essere rappresentato, infatti, sia dall’utente finale che utilizza il sistema (v.di Figura 12), sia dall’utente in quanto detentore di un account presso un server FTP (v.di Figura 13). Nel mondo reale un unico utente finale può possedere più account utente distinti.

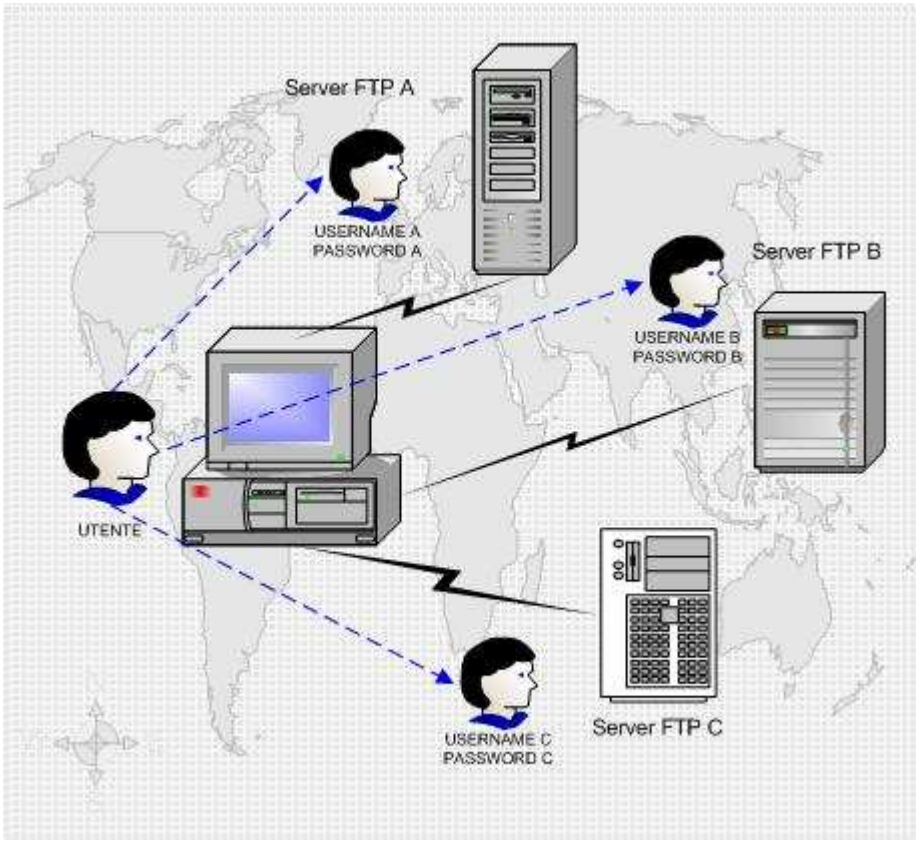

Figura 12 - Diverse identificazioni lato server per uno stesso utente lato client

I componenti progettati sono strettamente dedicati all'interazione tra un server FTP e l'infrastruttura TuCSoN, cosa che fa propendere per la scelta della seconda possibilità.

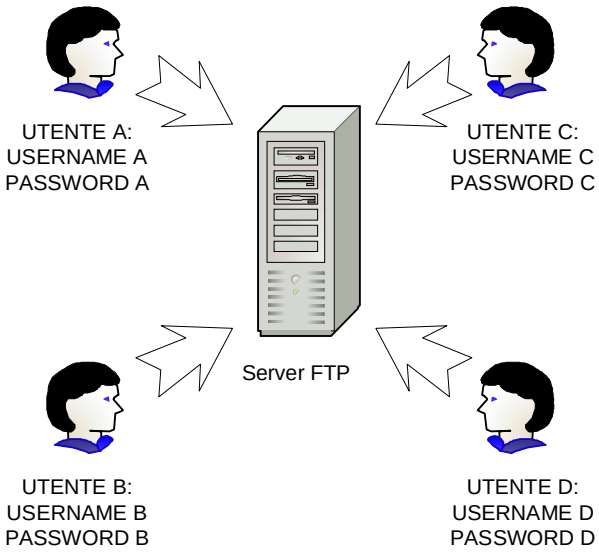

Figura 13 - L'utente lato server

Una gestione utenti di più alto livello è demandata ad un eventuale applicativo che faccia uso dei componenti in questione. Le possibilità implementative più significative sono molteplici:

- Una prima realizzazione più semplice potrebbe essere quella di prevedere un unico spazio per raccogliere le tuple rappresentative dei trasferimenti richiesti da parte tutti gli utenti; questa soluzione ha però il difetto che le dimensioni dei centri di tuple potrebbero diventare molto elevate, al punto da causare una decadenza delle prestazioni.
- Un'altra possibilità consiste nel prevedere un centro di tuple per ogni utente, contraddistinto e identificato dal nome utente; in questo caso il nome o l'indirizzo del server FTP a cui collegarsi deve essere contenuto nelle singole tuple che rappresentano i trasferimenti. Identificando il

centro di tuple con il solo nome utente, però, c'è lo svantaggio che due utenti diversi potrebbero vedere condivise le tuple nello stesso centro di tuple; nel mondo reale, infatti, due persone diverse possono avere gli stessi *user name* su due server distinti. Questa intrinseca disomogeneità porta a scartare questa possibilità.

- Una terza alternativa, simile alla precedente, può essere quella di prevedere un centro di tuple per ogni utente, ma contraddistinguerlo con *user name* e *server* (nome o indirizzo). In questo modo si garantiscono omogeneità e ordine. Le singole tuple, che rappresentano i trasferimenti di file, possono non contenere informazione sul server in quanto già contenuta nel nome del centro di tuple specifico.

Come ulteriore estensione si potrebbe pensare all'utilizzo di ulteriori centri di tuple per tenere traccia di ulteriori informazioni, ad esempio prevedere un centro di tuple in cui memorizzare i trasferimenti effettuati correttamente o i trasferimenti falliti. In questi spazi ulteriori si potrebbero poi immettere copie delle tuple di trasferimento, le tuple di trasferimento stesse o tuple con un diverso schema.

La soluzione adottata nel progetto sviluppato in questa tesi prevede tre centri di tuple per ogni utente e server: uno contiene le richieste di trasferimento accodate, un secondo contiene i

trasferimenti effettuati, l'ultimo, infine, gli errori di trasferimento. E' stato scelto di identificare il server mediante l'indirizzo IP, e non mediante il nome, in quanto non tutti i server FTP possiedono un nome registrato, mentre tutti possiedono obbligatoriamente un indirizzo di rete.

I nomi dei centri di tuple hanno dunque il seguente formato:

u + ***user name formattato*** **+ s** + ***indirizzo IP del server formattato*** +

rispettivamente:

- **Upload**
- **Done**
- **Error**

E' necessario formattare opportunamente la stringa IP del server evitando l'uso di punti, sostituendoli ad esempio con il carattere '_'. Gli indirizzi formattati sono del tipo 127_0_0_1. Lo stesso vale per lo user name.

Le tuple utilizzate nel progetto sono di tre tipi diversi, distinte in base alla condizione del trasferimento e quindi alle informazioni di cui si deve tenere traccia:

- Accodamento:

  ftpUpload(nome file locale, path locale, nome file remoto, path remoto, priorità, data e ora di ingresso in coda)

- Trasferimento effettuato:

ftpDone(nome file locale, path locale, nome file remoto, path remoto, priorità, data e ora di ingresso in coda, data e ora di inizio trasferimento, data e ora di fine trasferimento)

- Errori nel trasferimento:

  ftpError(nome file locale, path locale, nome file remoto, path remoto, priorità, data e ora di ingresso in coda, data e ora di inizio trasferimento, data e ora di errore)

5. PROGETTO DEI COMPONENTI

5.1 Java e TuCSoN

Per implementare i componenti è stato scelto il linguaggio Java, oltre che per la sua portabilità, prerogativa fondamentale in un sistema distribuito, anche per il fatto che TuCSoN mette a disposizione alcuni package per accedere a basso livello all'infrastruttura, e costruire agenti e applicazioni che la utilizzano. In particolare i due package utilizzati nel progetto sono:

- `alice.tucson.api`, che contiene le classi che descrivono e implementano l'astrazione dell'infrastruttura, i contesti di coordinazione, gli agenti, i centri di tuple.
- `alice.logictuple`, che contiene le classi che permettono di costruire e gestire le tuple logiche e i loro argomenti.

In generale, per interagire con l'infrastruttura da Java, è necessario acquisire un contesto di coordinazione, attraverso il quale si possono poi eseguire sui centri di tuple le primitive descritte in precedenza.

5.2 Gli agenti

Il prototipo realizzato è costituito da alcuni package, che verranno analizzati nel dettaglio in seguito. I componenti più importanti su cui è necessario focalizzare inizialmente l'attenzione sono, infatti, gli agenti **FtpInsertUploadAgent** e

FtpUploadAgent. Il primo si occupa di trasformare una richiesta di upload di un file ad un dato server in una tupla logica e di inserire quest'ultima nel centro di tuple opportuno. Il secondo recupera le tuple dal centro di tuple relativo ad un dato utente e, dopo averle interpretate opportunamente prelevando le informazioni necessarie alla connessione, gestisce l'interfacciamento con il server FTP e l'upload dei file vero e proprio. I due agenti possiedono tutte le caratteristiche per essere parte dell'*Application Logic* di un programma che li utilizzi.

5.2.1 FtpInsertUploadAgent (v.di Figura 14)

FtpInsertUploadAgent contiene un oggetto di tipo *FtpAutentication*, che rappresenta una autenticazione di un utente su un dato server; questa, associata ad una interfaccia generica che rappresenta un trasferimento, *ITucsonFtpMultipleTransfer*, è in grado di fornire all'agente tutte le informazioni e i metodi di cui ha bisogno per lavorare. *ITucsonFtpMultipleTransfer* raggruppa i metodi delle interfacce *IFtpMultipleTransfer*, che modella un trasferimento di file, e *ITucsonConverter*, che fornisce i metodi che deve avere un generico convertitore da oggetto java a tupla logica. *ITucsonFtpMultipleTransfer* viene associata dal costruttore alla classe *TucsonFtpMultipleTransfer* che la implementa. Attraverso i metodi dell'interfaccia *IFtpMultipleTransfer* è possibile modificare gli attributi relativi al trasferimento, mentre attraverso i metodi di *ITucsonConverter* è possibile generare le tuple logiche che verranno poi inserite nei

centri di tuple. I nomi da utilizzare per i centri di tuple, vengono reperiti passando l'oggetto *FtpAutentication* della classe a metodi statici di FtpUploadAgent, che vengono mantenuti solo in tale classe per garantire uniformità.

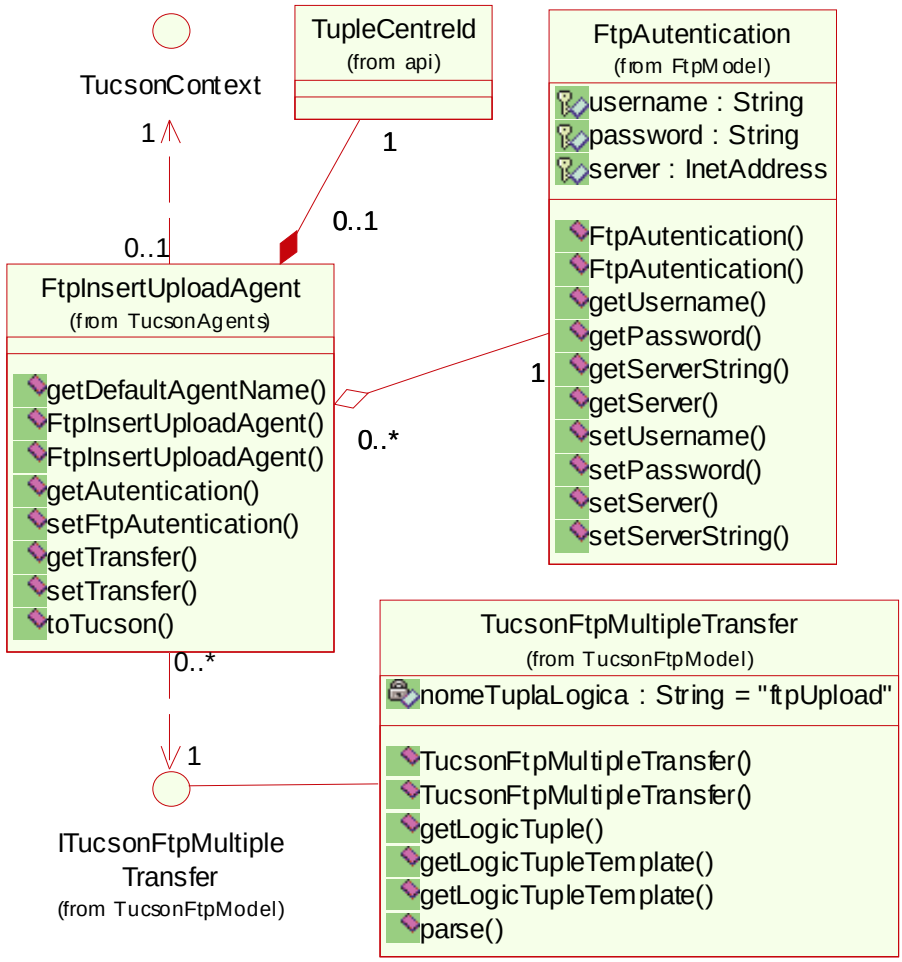

Figura 14 - Diagramma UML della classe FtpInsertUploadAgent

5.2.2 FtpUploadAgent (v.di Figura 15)

Contiene un oggetto di tipo *FtpAutentication,* e tre interfacce, *ITucsonFtpMultipleTransfer*, *ITucsonFtpMultipleTransferDone*, *ITucsonFtpMultipleTransferError*; queste ultime due interfacce sono analoghe alla prima, ma rispettivamente relative ad un trasferimento effettuato e ad un trasferimento interrotto a causa di errori. Oltre a questi componenti l'agente ha come attributi: una flag che specifica all'agente se tenere conto della priorità dei trasferimenti, un valore numerico che indica la porta per la connessione di controllo FTP verso il server, infine un oggetto di tipo *FtpClient*, ovvero l'implementazione di un client FTP. L'agente, durante la sua esecuzione, effettua la connessione al server FTP attraverso il componente *FtpClient*, e si autentica al server utilizzando i dati forniti dall'oggetto *FtpAutentication*; se la connessione è stabilita, estrae tuple "di tipo" *ITucsonFtpMultipleTransfer* dallo spazio di tuple opportuno, le interpreta e inizia il trasferimento del file corrente. A seconda del risultato della buona riuscita o meno del trasferimento l'agente trasforma le tuple in tuple di tipo *ITucsonFtpMultipleTransferDone* o di tipo *ITucsonFtpMultipleTransferError* e le sposta nel centro di tuple appropriato (trasferimenti effettuati o errori). Da notare che l'agente è di fatto un thread in continua esecuzione che, dal momento in cui viene avviato, tenta di reperire tuple ed effettuare trasferimenti. Per impedire che questo loop continuo causi inefficienza, nell'implementare l'agente sono state utilizzate le

primitive di lettura bloccanti; in tal modo, in caso tupla non presente, la sua esecuzione viene sospesa dallo scheduler della JVM (*Java Virtual Machine*). L'esecuzione potrà riprendere quando verrà inserita almeno una tupla. Nella condizione di blocco l'agente termina anche la connessione con il server ftp, per evitare lo spreco di risorse. Come ultima osservazione, una nota sulla priorità: nel caso in cui la modalità di trasferimento sia "con priorità", il comportamento dell'agente è non bloccante, ovvero, se ad esempio non sono presenti trasferimenti ad alta priorità l'agente passerà direttamente al prelievo delle tuple con priorità media; lo stesso vale per i trasferimenti a priorità bassa.

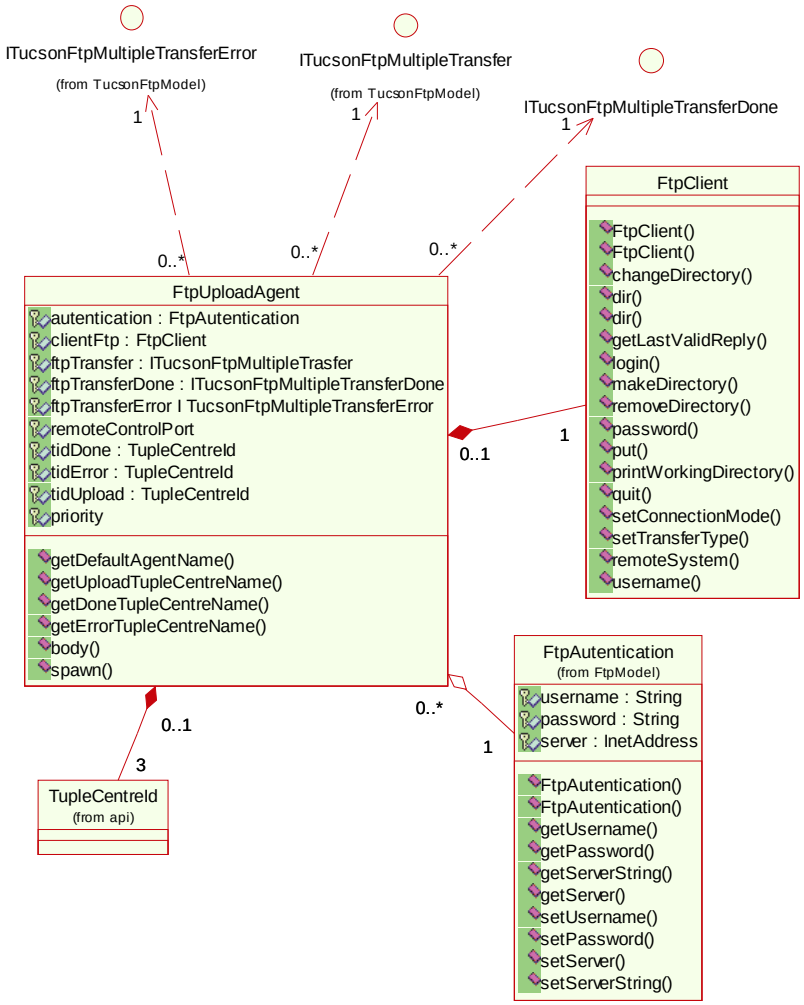

Figura 15 - Diagramma UML della classe FtpUploadAgent

5.3 Il prototipo

5.3.1 I package

Il prototipo si compone di alcuni package:

- **tucsonagents**: contiene i due agenti precedentemente descritti, *FtpInsertUploadAgent* e *FtpUploadAgent*, e un

terzo agente *RdAllAgent*, che si occupa di reperire tutte le tuple di un certo tipo da un centro di tuple.

- **ftpmodel**: contiene classi e interfacce che descrivono il modello FTP (v.di Figura 17). Da notare la classe *FtpAutentication* (v.di Figura 16), che modella una autenticazione ad un server FTP e si occupa della risoluzione degli eventuali nomi di server inseriti, in indirizzi.
- **ftp**: implementa un client FTP, ed è il package necessario all'interfacciamento con il server remoto (v. di par. 5.4).

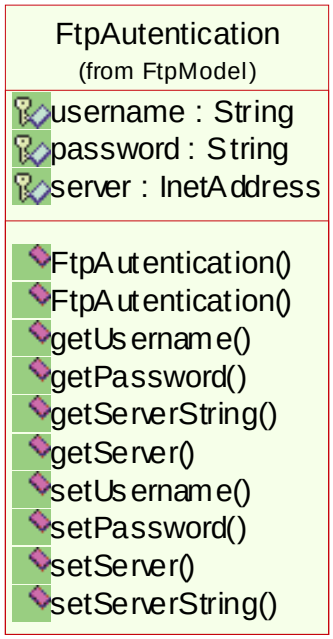

Figura 16 - La classe FtpAutentication

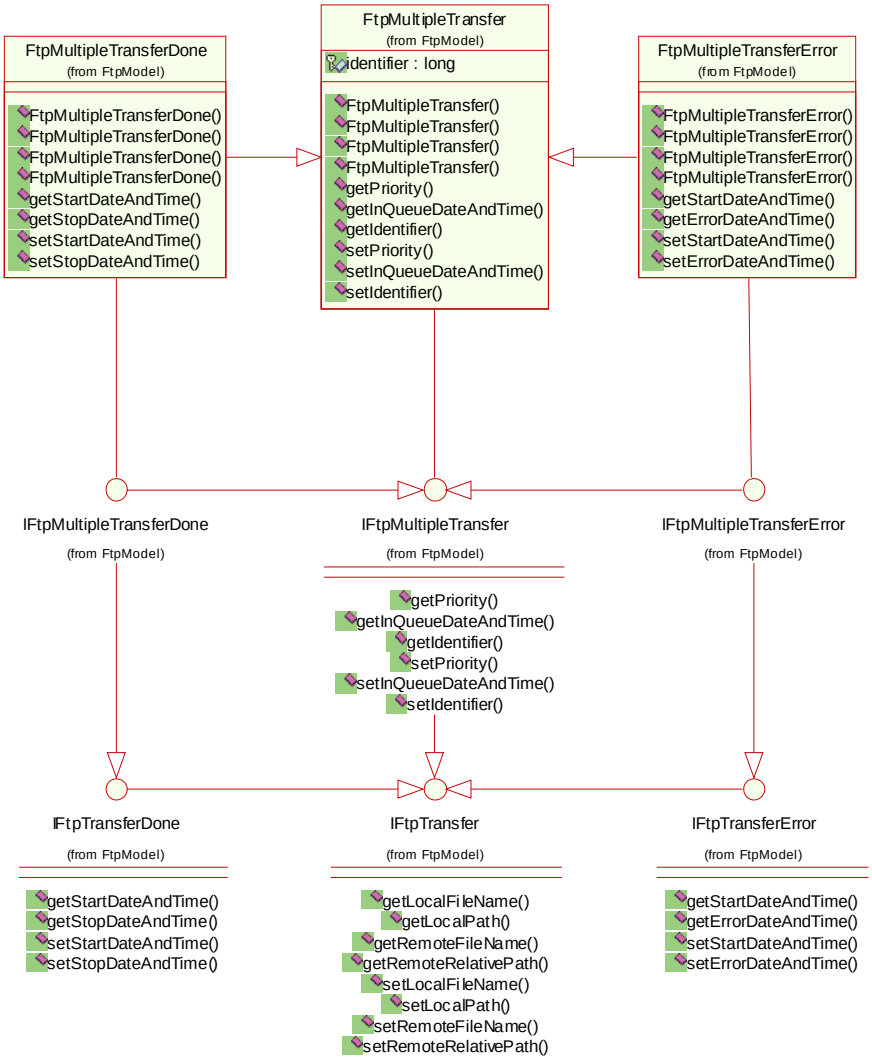

Figura 17 - Diagramma UML del package ftpmodel

Figura 18 - Diagramma UML del package ftp

- **tucsonmodel**: contiene l'interfaccia *ITucsonConverter*. Le classi che implementano questa interfaccia incapsulano metodi che permettono di ottenere una tupla logica, o template di essa, a partire dagli attributi della classe e di inizializzare gli attributi a partire da una tupla logica corretta.
- **tucsonftpmodel**: contiene classi e interfacce che uniscono le caratteristiche di quelle del modello FTP con

l'interfaccia del modello TuCSoN, e sono pertanto i componenti utilizzati dagli agenti realizzati.

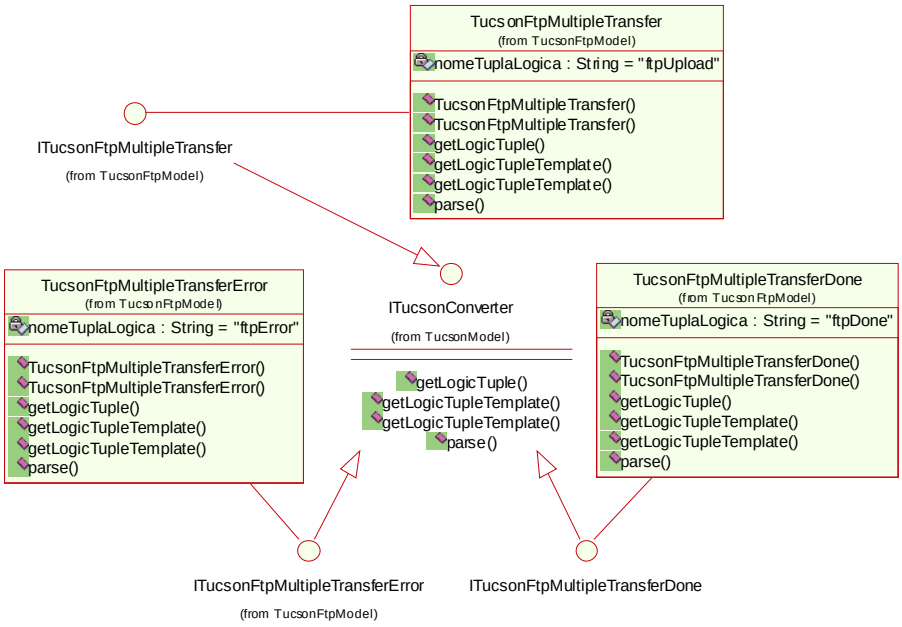

Figura 19 - Diagramma UML del package tucsonftpmodel

- **guicommons**: contiene alcune classi utili, comuni a tutti i componenti grafici realizzati.
- **test**: contiene le interfacce grafiche realizzate per i due agenti; in particolare: *FtpInsertUploadAgentGui*, *IUAAutenticationPanel*, *IUATransferPanel* sono le classi utilizzate per interagire con l'agente *FtpInsertUploadAgent*, mentre *FtpUploadAgentGui* e *UAMainPanel* sono le classi utilizzate per interagire con l'agente *FtpUploadAgent*. Esiste anche un'ulteriore classe, *FtpUploadAgentCL*, che

permette di utilizzare l'agente *FtpUploadAgent* da linea di comando.

5.3.2 I componenti grafici

Per collaudare i componenti attraverso una applicazione più *user friendly*, dunque, è stata sviluppata in maniera prototipale anche la parte di *presentation logic* per i due componenti. Per garantire modularità in essa è incapsulata solo la *GUI* (Graphical User Interface) e gestita l'immediata validazione dei dati immessi dall'utente. Attraverso semplici riferimenti, dal *presentation logic* è poi possibile istanziare gli agenti, inizializzarli, e utilizzarli. Le due classi che contengono il *main*, e dunque "eseguibili", sono: *FtpInsertUploadAgentGui* e *FtpUploadAgentGui*.

In particolare *FtpInsertUploadAgentGui* si occupa di creare il frame principale dell'applicazione, e di inserire in esso un'istanza della classe *IUAAutenticationPanel*, che contiene i componenti grafici per l'autenticazione, e mantiene un riferimento ad un'istanza di *FtpAutentication*. L'*IUAAutenticationPanel*, ad autenticazione avvenuta, carica poi (mediante *OkListener*) nel frame un'istanza di *IUATransferPanel,* che, oltre ad istanziare l'agente *FtpInsertUploadAgent* passandogli un riferimento all'*FtpAutentication* ricevuto dall'*IUAAutenticationPanel*, contiene i componenti grafici per inserire un trasferimento nel centro di tuple. In particolare la classe *InsertListener* estrae dai

componenti dell'interfaccia i dati per il trasferimento e li passa all'agente, chiamando in seguito il metodo *toTucson()* di quest'ultimo, che provvede all'effettivo trasferimento verso l'infrastruttura. Attraverso questa applicazione è anche possibile visualizzare i contenuti dei tre centri di tuple relativi all'utente autenticato.

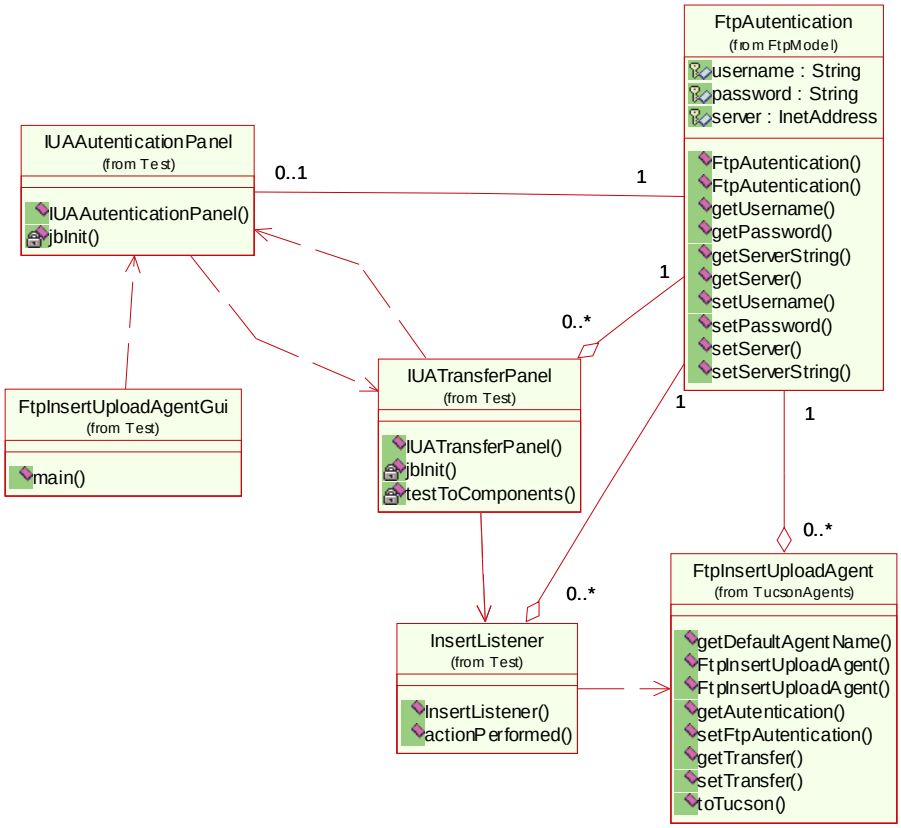

Figura 20 – L'interfaccia grafica dell'agente FtpInsertUploadAgent

Attraverso l'interfaccia di questo componente è anche possibile visualizzare i contenuti dei centri di tuple, istanziando un oggetto

di tipo *TupleCenterPanel*, che si appoggia a sua volta all'agente *RdAllAgent*. Quest'ultimo è infatti in grado di reperire da un centro di tuple tutte le tuple che soddisfano un certo schema.

FtpUploadAgentGui si occupa di creare il frame principale dell'applicazione e di inserire in esso un'istanza della classe *UAMainPanel*; quest'ultima contiene i componenti grafici per l'autenticazione, per impostare i dati relativi alla connessione FTP, e per decidere la modalità di trasferimento. Se i campi sono tutti compilati, l'agente è perfettamente in grado di svolgere il suo compito; la classe *StartListener*, che attende gli eventi di *click* del mouse sul pulsante di start, istanzia, inizializza opportunamente a partire dai campi e avvia l'agente *FtpUploadAgent*.

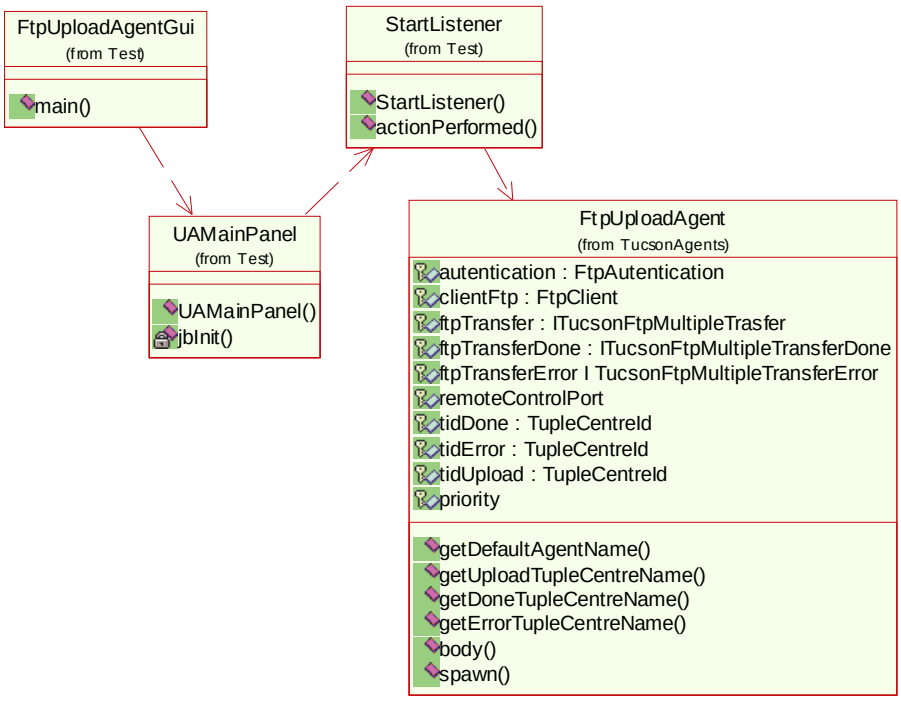

Figura 21 – L'interfaccia grafica dell'agente UploadAgent

5.3.3 **FtpUploadAgentCL**

La classe *FtpUploadAgentCL* svolge gli stessi compiti della classe *FtpUploadAgentGui*, ovvero istanziare, inizializzare e avviare un agente di upload. Invece di richiedere l'interazione diretta dell'utente sui componenti grafici, però, è possibile eseguire questa classe passando tutti i parametri necessari da linea di comando, il che rende il componente facilmente utilizzabile all'interno di procedure automatizzate.

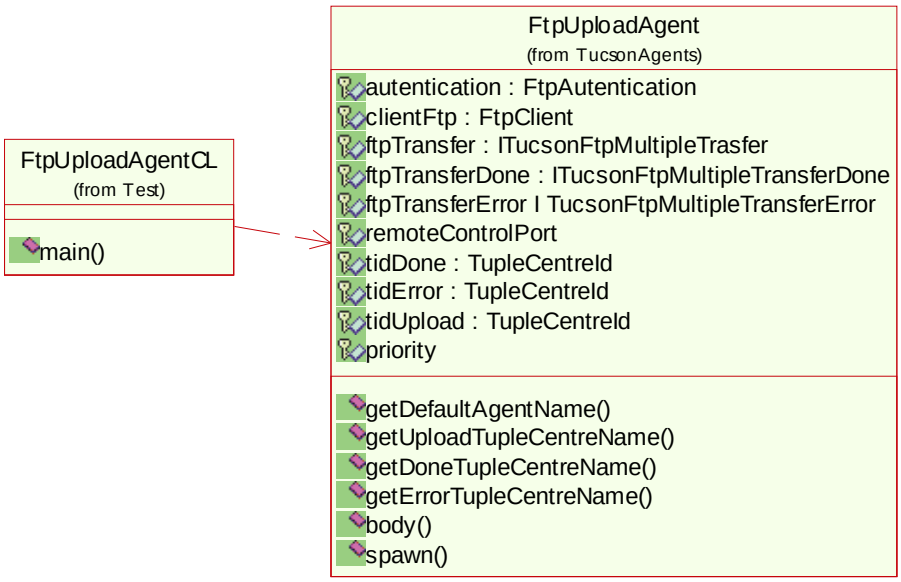

Figura 22 - Il main istanzia e avvia l'agente FtpUploadAgent

5.4 Utilizzo di componenti preesistenti

Nel progetto affrontato è necessario implementare un componente o un insieme di componenti su cui l'agente *FtpUploadAgent* possa appoggiarsi per interfacciarsi con il server FTP.

Java non prevede per l'FTP un package o una libreria contenenti classi che incapsulino il protocollo. D'altra parte Sun fornisce un package contenente un insieme di classi che implementano l'FTP, ma il loro funzionamento è strettamente legato al sistema operativo, il che impedisce la portabilità delle applicazioni.

Sul web, segnalata dal sito della Sun, è però disponibile una libreria freeware, interamente scritta in Java puro, denominata

Java FTP client; essa contiene un componente che realizza un client FTP di base, ed è totalmente compatibile con lo standard RFC 959. Queste caratteristiche hanno permesso di sfruttare la libreria per la realizzazione di buona parte dei componenti in questione.

In particolare la libreria contiene il package *com.enterprisedt.net.ftp*, del quale sono state utilizzate le seguenti classi:

- *FTPClient*: implementa il protocollo FTP dal lato client; possiede metodi che realizzano le operazioni FTP più comuni.
- *FTPConnectMode*: permette di specificare al server, secondo lo standard FTP, il ruolo che deve svolgere nella comunicazione (attivo, passivo).
- *FTPTransferType*: permette di specificare il tipo di file che si vuole inviare o ricevere sulla connessione (testuale ASCII, o binario).
- *FTPReply*: è una classe che incapsula le risposte FTP del server, e possiede metodi per interpretarle.
- *FTPException*: è l'*eccezione* generata dalle classi precedenti quando qualche operazione FTP fallisce.

5.4.1 Il package ftp

Per permettere migliorie e diverse implementazioni future del protocollo, i componenti descritti nel precedente paragrafo sono stati incapsulati in una serie di classi wrapper, contenute nel

package **ftp**, il cui unico scopo è quello di fornire metodi standard al "mondo esterno", che non fanno altro che invocare a loro volta i metodi delle classi contenute.

FtpClient, ad esempio, che incapsula *FTPClient*, possiede tutti i metodi necessari per un trasferimento di file verso un server FTP (estendibile facilmente al *download*), e sono quelli effettivamente utilizzati dall'*FtpUploadAgent*. I metodi hanno nomi del tutto somiglianti ai comandi FTP.

La seguente figura mostra la relazione tra le due classi, e l'elenco dei metodi incapsulati dalla classe *FtpClient*.

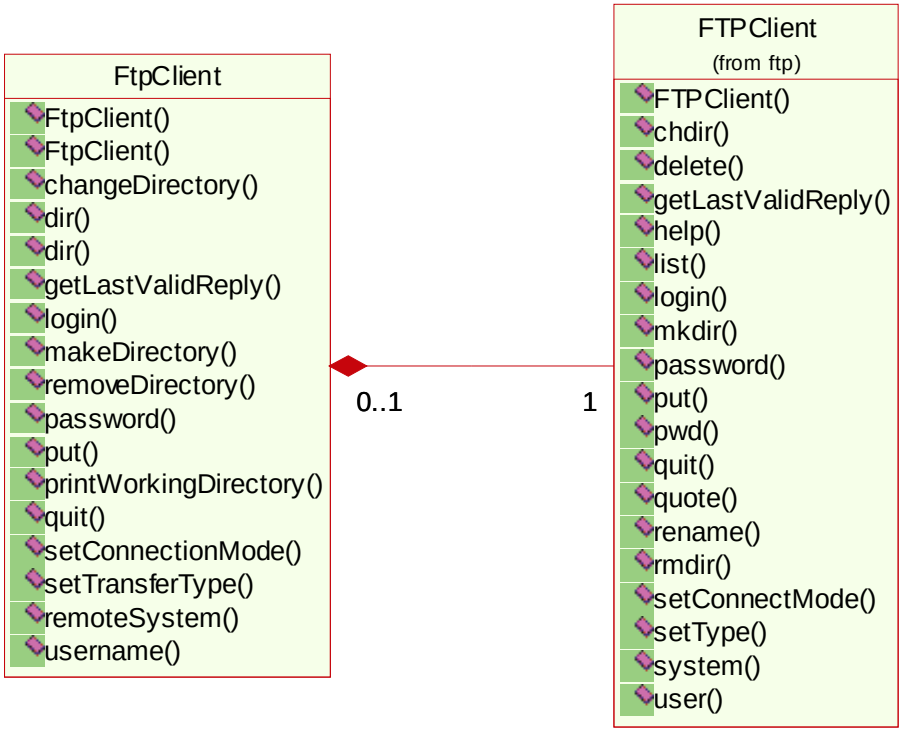

Figura 23 - La relazione tra la classe FtpClient e la classe FTPClient

6. COLLAUDO DEL SISTEMA

Per collaudare i componenti e verificare le scelte progettuali è opportuno simulare l'architettura completa del sistema. Ciò richiede l'utilizzo di un ambiente in cui sia installato un nodo TuCSoN e, per il test in locale, che sia presente un server FTP. Il test dei componenti è stato effettuato in ambiente Windows 2000, con piattaforma JDK (*Java Development Kit*) 1.4 Standard Edition, Tucson 1.3.0; per il test in locale come server FTP è stato scelto *BulletProof FTP Server 2.15*, sia per la sua interfaccia *user friendly* molto ben realizzata, sia per la completa compatibilità con lo standard RFC 959.

6.1 Ftp server

6.1.1 Configurazione

Per poter utilizzare *BulletProof* è necessario creare gli accounts per gli utenti che si desidera possano accedere ad una zona del file system locale. La seguente figura mostra la finestra dalla quale è possibile gestire gli accounts.

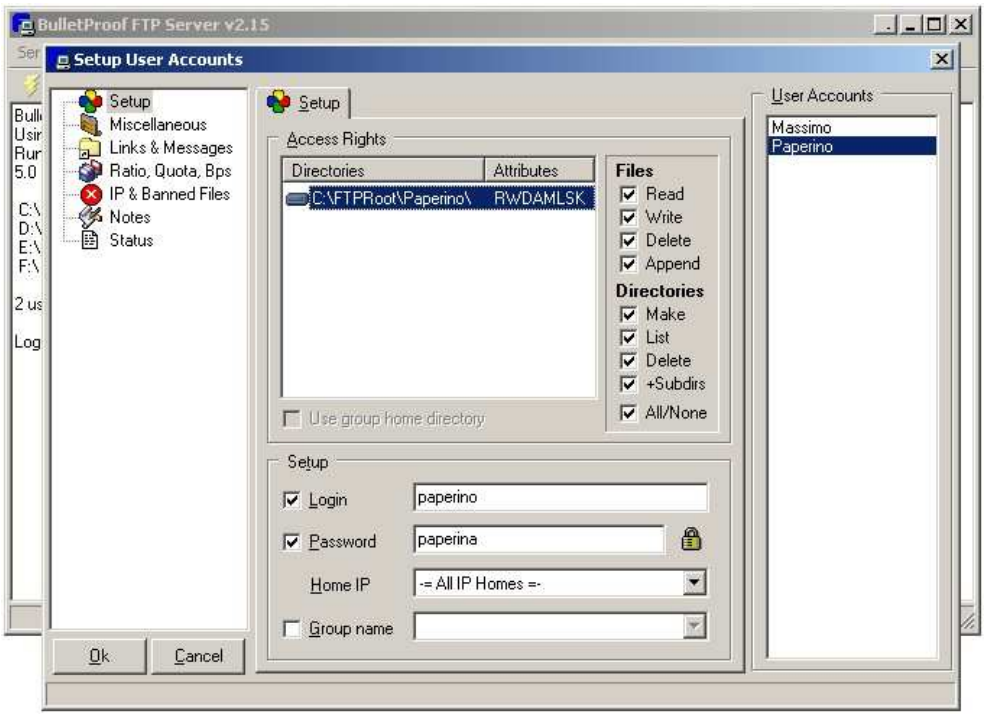

Figura 24 - Gestione degli utenti in BulletProof

Dalla figura si nota, ad esempio, che l'utente *Paperino* ha controllo completo sulla directory *C:\FTPRoot\Paperino*, alla quale accede autenticandosi al server con username *paperino* e password *paperina*. Cliccando sul riquadro "User Accounts" si possono creare nuovi utenti; cliccando sul riquadro in cui sono contenute le directory, si possono dare all'utente diritti su altre cartelle.

6.1.2 Utilizzo

Le opzioni di programma disponibili sono molteplici: è possibile impostare il numero della porta su cui avviare il server, il numero massimo di utenti accettato, opzioni generali per la gestione degli

utenti, opzioni di visualizzazione, opzioni per lo scheduling di alcuni task, e molte altre funzioni. E' possibile inoltre creare un file di log, monitorare l'attività del server, visualizzare statistiche e informazioni sugli utenti connessi.

Dopo aver creato tutti gli utenti/gruppi di utenti desiderati e configurato l'applicazione è poi possibile avviare il server, che da questo istante rimarrà in attesa di nuove connessioni. L'interfaccia grafica del server è mostrata in figura.

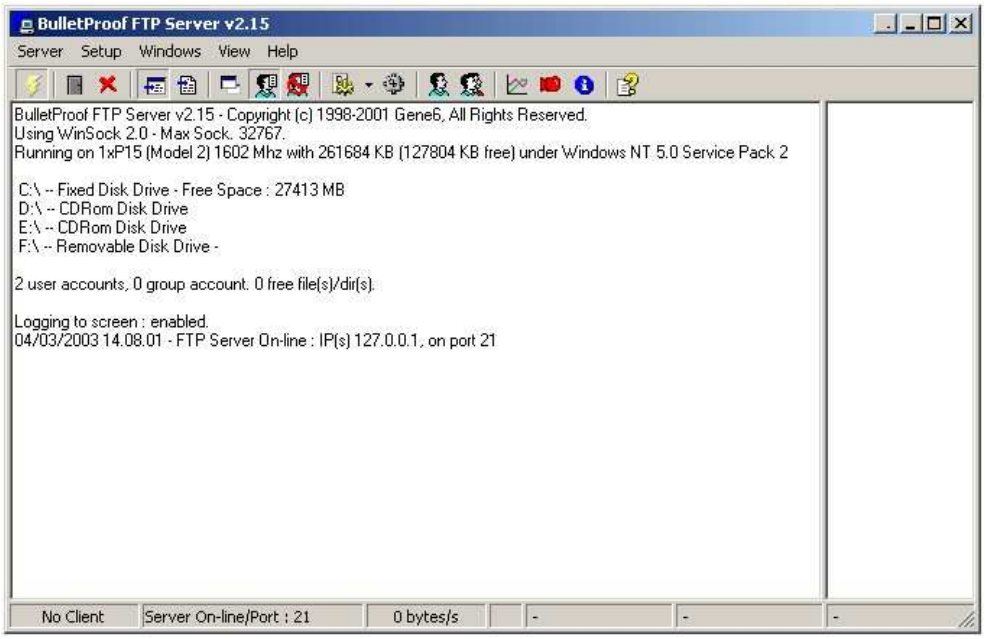

Figura 25 - Avvio del server BulletProof

6.2 Configurazione di TuCSoN

Il passo successivo consiste nell'attivazione, su uno degli host disponibili, di un nodo TuCSoN. A questo scopo, aprire la

sottocartella *bin* nella directory in cui è installato TuCSoN, ed eseguire il file *Node.bat* per avviare il nodo TuCSoN.

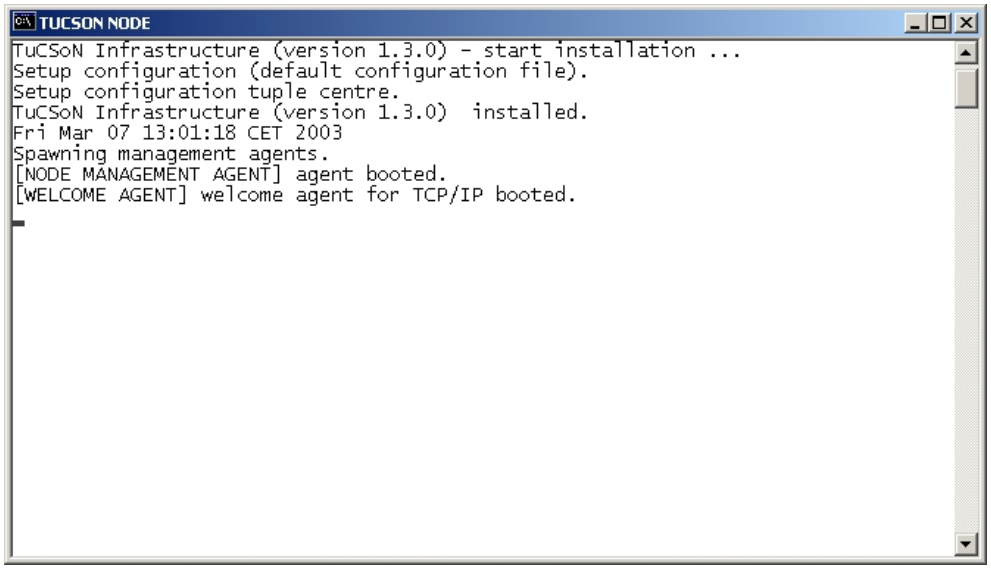

Figura 26 - Avvio del nodo TuCSoN

6.3 Test dei componenti

A questo punto, una volta attivo il nodo TuCSoN, è possibile testare gli agenti realizzati, lanciando *FtpInsertUploadAgentGui*, *FtpUploadAgentGui, FtpUploadAgentCL.* Per l'esecuzione dei componenti è necessario includere nel *classpath* della JVM le librerie di Tucson (*tucson.jar* e *tucson-full.jar*) per tutti e tre i componenti, e la libreria del Client FTP (*ftp.jar*) per *FtpUploadAgentGUI* e *FtpUploadAgentCL*.

Per eseguire *FtpUploadAgentGui*, ad esempio, è necessario procedere come segue:

- posizionarsi nella cartella superiore a "Test";

- eseguire *java –cp <percorso tucson.jar>;<percorso tucson-full.jar>;<percorso ftp.jar>; Test.FtpUploadAgentGui*

6.3.1 FtpInsertUploadGui

Eseguendo il componente apparirà una finestra di autenticazione, che richiede di inserire i parametri necessari al login su un dato server ftp (v.di Figura 27).

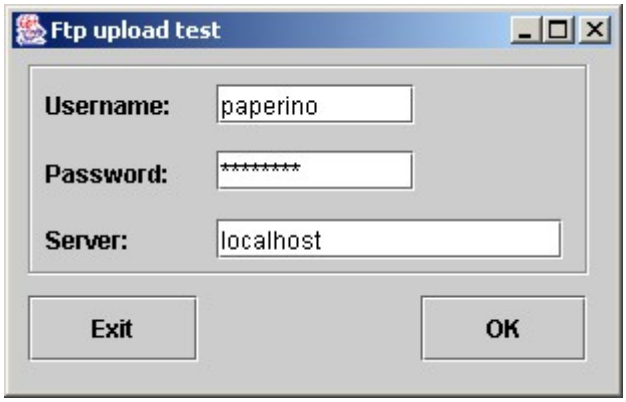

Figura 27 - Finestra di autenticazione di FtpInsertUploadGui

Nel caso in cui l'utente immetta un nome di server che non può essere risolto, il sistema non permetterà di proseguire (v.di Figura 28).

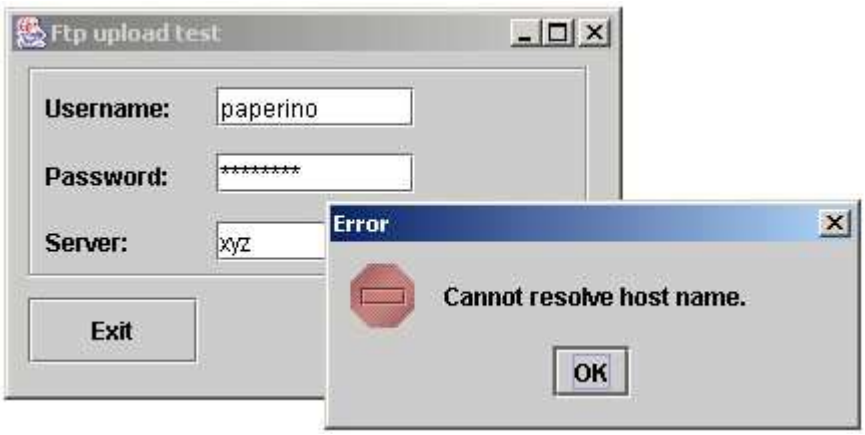

Figura 28 - Errore di autenticazione

Se, al contrario, l'autenticazione avviene correttamente, apparirà la finestra mediante la quale è possibile inserire i dati relativi al trasferimento (v.di Figura 29).

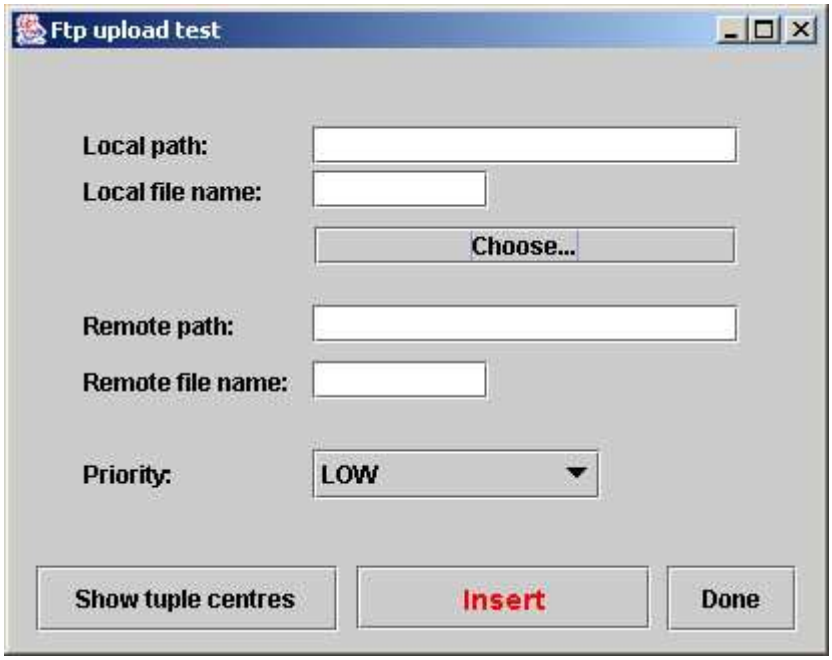

Figura 29 - Finestra per l'inserimento richieste

Cliccando sul pulsante “choose” è possibile reperire il file da trasferire mediante una finestra di dialogo (v.di Figura 30).

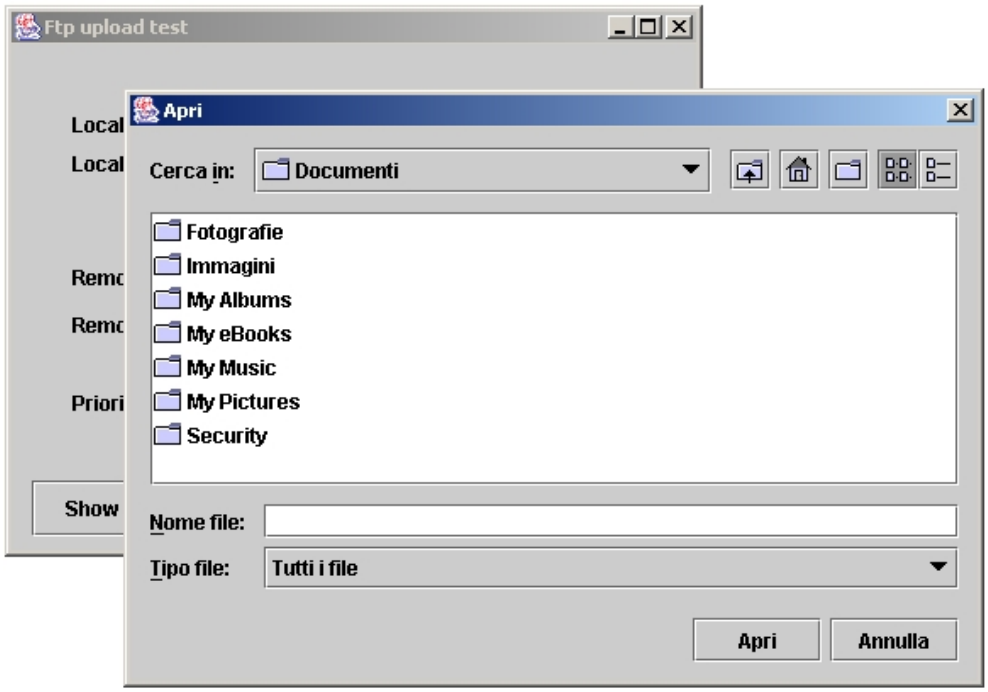

Figura 30 - Finestra di dialogo

Nel campo “Remote file name” l’utente dovrà inserire il nome che vuole dare al file in remoto, mentre nel campo “Remote path” dovrà inserire l’eventuale percorso (relativo alla cartella di root in remoto) del file in remoto. Mediante il combo box “Priority” è possibile settare una priorità per la tupla; il default è LOW.

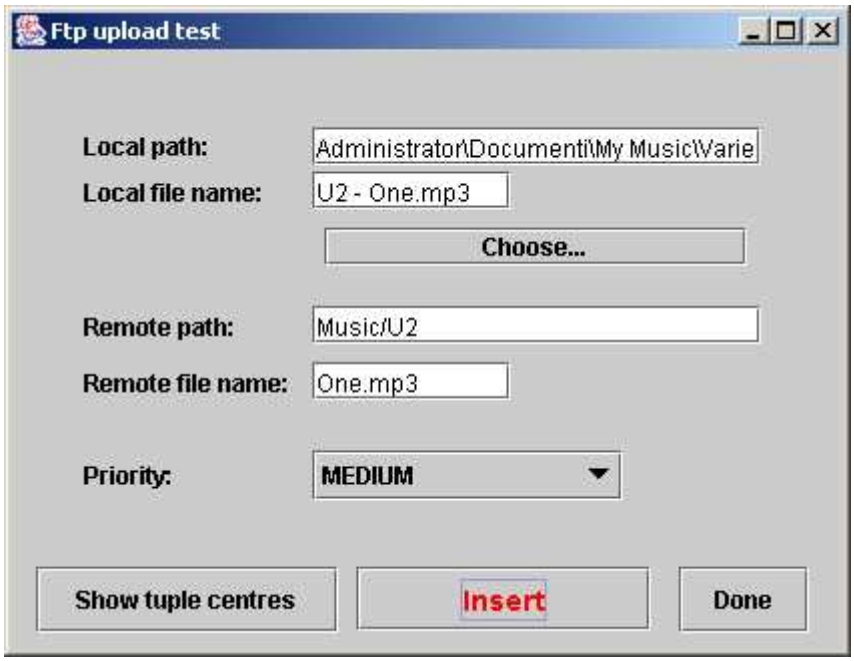

Figura 31 - Fase di inserimento dei dati

A questo punto è possibile procedere all'inserimento delle richieste; se non è stato avviato un nodo TuCSoN l'utente verrà avvisato attraverso una finestra di dialogo (v.di Figura 32).

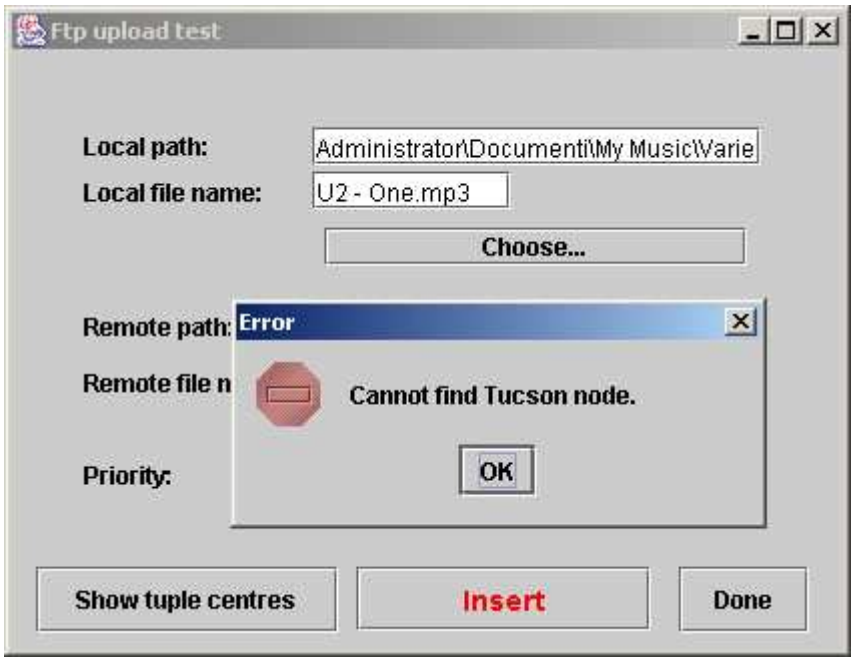

Figura 32 - Errore di nodo TuCSoN non trovato

Mediante il pulsante “Show tuple centres” è possibile visualizzare il contenuto dei tre centri di tuple (v.di Figura 33).

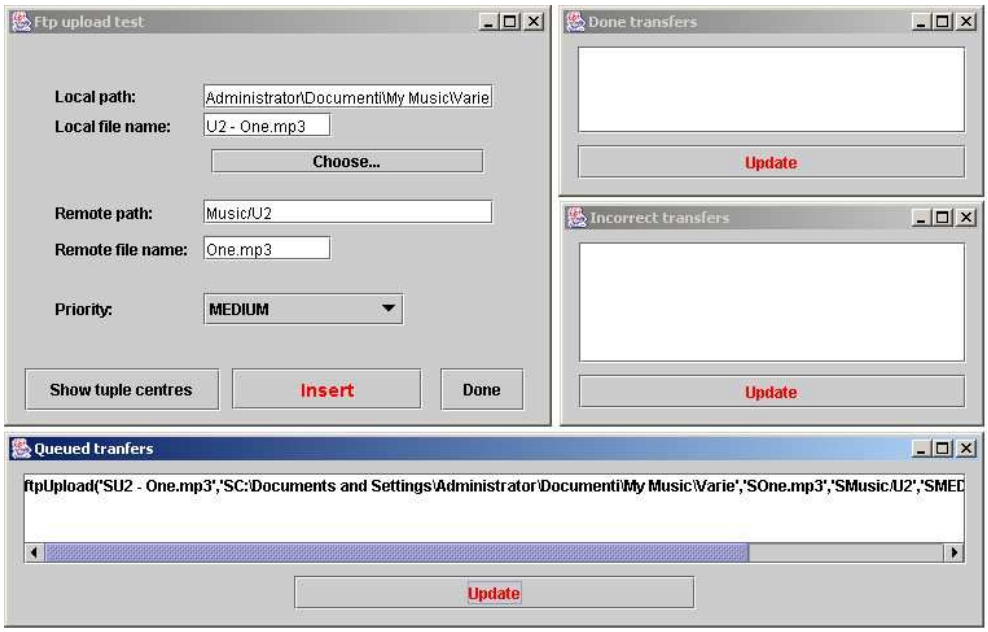

Figura 33 - Visualizzazione del contenuto dei centri di tuple

6.3.2 FtpUploadAgentGui

Figura 34 - Interfaccia grafica di FtpUploadAgent

La Gui dell'agente *FtpUploadAgent* (v.di Figura 34) presenta un'unica finestra che, oltre ai campi dell'autenticazione, presenta una casella di testo in cui specificare il numero della porta su cui risiede il servizio sull'host sopra specificato, un campo "Use priority" mediante il quale si può comunicare all'agente se trasferire le tuple tenendo conto della priorità. Mediante il pulsante di Start viene avviato l'agente, che proseguirà la sua esecuzione in background; nel caso in cui l'agente non riesca a stabilire la connessione con il Server FTP o con il nodo TuCSoN, l'utente viene avvisato mediante una finestra di dialogo. Nel caso in cui durante l'attività dell'agente venga interrotta la connessione FTP o la comunicazione con il nodo TuCSoN, l'agente termina la sua esecuzione.

Cliccando sul pulsante di start si avvia il trasferimento del file inserito mediante l'agente *FtpInsertUploadAgent*. Informazioni sullo stato del trasferimento si possono ottenere sia dalla finestra di standard out del componente, sia dalla finestra principale di BulletProof (v.di Figura 35).

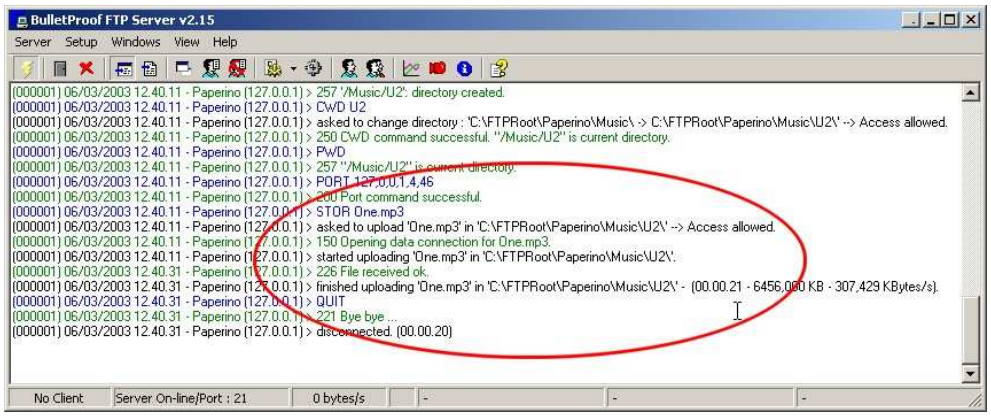

Figura 35 - Finestra di log di BulletProof

6.3.3 FtpUploadAgentCL

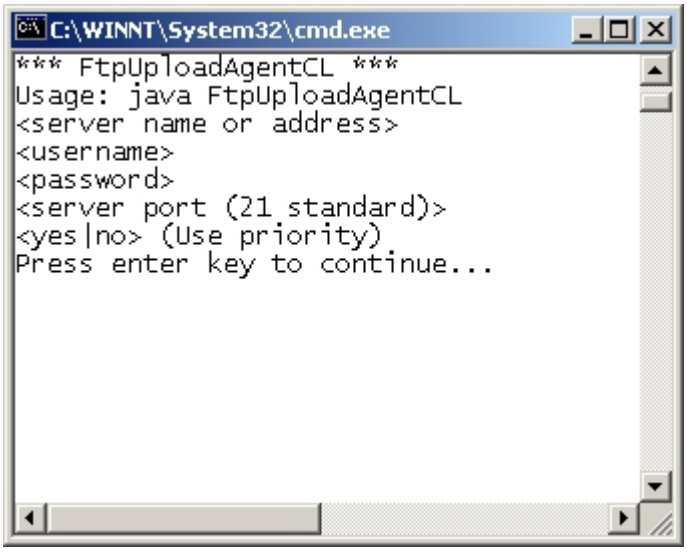

Figura 36 - Interfaccia da linea di comando di FtpUploadAgentCL

FtpUploadAgentCL svolge gli stessi compiti di *FtpUploadAgentGui*, ma presenta un'interfaccia da linea di

comando che favorisce l'uso in automatico dell'agente *FtpUploadAgent* (v.di Figura 36).

I parametri che l'utente deve fornire al componente sono:

- nome o indirizzo del server
- nome utente
- password
- porta per il servizio FTP
- utilizzo della priorità.

6.3.4 Utilizzo di FtpUploadAgent in remoto

Per completare il quadro relativo all'architettura del sistema si è effettuato il test dei componenti appoggiandosi ad un server FTP remoto su cui si disponeva di un account.

Sullo standard out dell'applicazione è possibile leggere le informazioni relative allo stato dell'interazione, come mostrato dalla seguente figura.

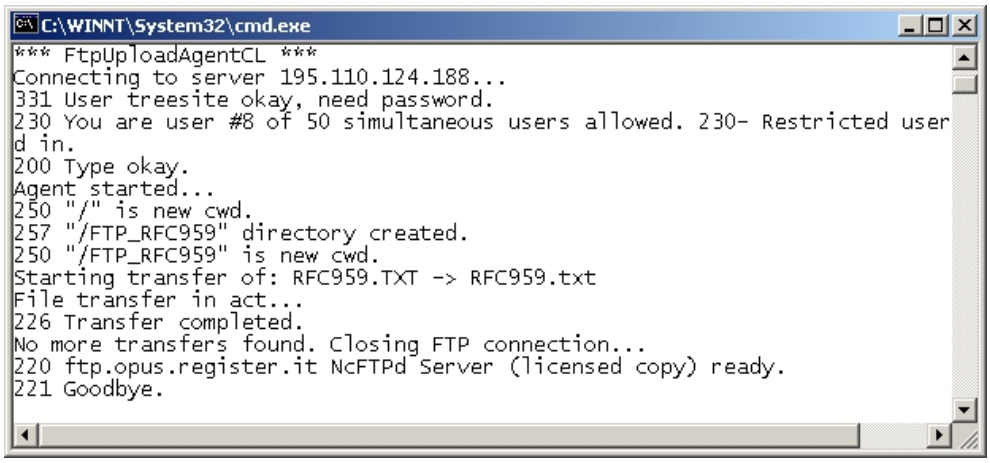

Figura 37 - Utilizzo in remoto dell'agente FtpUploadAgent

7. CONCLUSIONE

Questa nuova architettura permette di rendere “intelligente” il sistema FTP tradizionale inserendo un livello intermedio tra client e server. Il livello intermedio è costituito, in questo caso, dall’infrastruttura di coordinazione TuCSoN, le cui astrazioni di coordinazione, i centri di tuple, svolgono qui un duplice ruolo:

- da un lato, permettono di rappresentare le informazioni relative ai trasferimenti FTP in forma di tuple logiche di opportuno formato, consentendo così di accedere a tali informazioni facilmente;
- dall’altro, programmando opportunamente il centro di tuple di ogni utente, è possibile definire politiche specifiche per ogni utente anche molto complesse e svolgere in questo modo alcuni compiti automaticamente. Ad esempio tutti i trasferimenti verso un certo server o una certa cartella potrebbero essere spostati altrove, oppure i file con un certo nome potrebbero essere rinominati prima di essere inviati, altri ad esempio cancellati; in generale, attraverso ReSpecT è possibile ottenere ogni politica che si desideri.

Per l’interazione dell’infrastruttura con l’utente, da un lato, e con l’FTP server, dall’altro, sono stati realizzati due agenti, *FtpInsertUploadAgent* ed *FtpUploadAgent* che si occupano rispettivamente, di inserire nel centro di tuple le richieste dell’utente di trasferimento di file verso un certo server FTP, e di

provvedere, dopo aver reperito le richieste dal centro di tuple opportuno, all'effettivo upload di file. Il progetto è facilmente estendibile anche al lato download.

L'architettura delineata è aperta verso future estensioni: in particolare, l'uso delle tuple logiche come formato per la rappresentazione delle informazioni, non essendo limitato ad alcuno schema predefinito, può essere adottato anche per un numero illimitato di altri servizi. Ciò rende possibile, potenzialmente, un'estensione dell'attuale architettura in modo da fornire una visione unificata delle informazioni ottenute da servizi diversi (ad esempio FTP, MAIL, NEWS, HTTP) aprendo così la prospettiva di una loro possibile integrazione, secondo modelli e schemi tutti da definire.

BIBLIOGRAFIA

Standard Ftp:

www.w3.org - World Wide Web Consortium

Java e Ftp:

www.sun.com - Sun

www.enterprisedt.com - Enterprise Distributed Technologies

TuCSoN:

www.lia.deis.unibo.it/Research/TuCSoN - Tucson Home Page

APPENDICE

Di seguito è riportato il codice java di tutti i package realizzati. Di particolare rilievo è il package *tucsonagents*, che contiene il codice degli agenti TuCSoN progettati.

Package ftp

```
package ftp;

public class FtpActiveConnectionMode extends
FtpConnectionMode {

  public String connectionMode(){
    return "Active";
  }

}
                                  ---
package ftp;

public class FtpAsciiTransferType extends FtpTransferType {

  public String transferType(){
    return "Ascii";
  }

}
                                  ---
package ftp;

public class FtpBinaryTransferType extends FtpTransferType {

  public String transferType(){
    return "Binary";
  }

}
                                  ---
package ftp;

import com.enterprisedt.net.ftp.*;
import java.net.*;
import java.io.*;

public class FtpClient {

  FTPClient f=null;
```

```
  public FtpClient(InetAddress remoteAddress, int
remoteControlPort) throws IOException, FtpException {
    try{
      f=new FTPClient(remoteAddress,remoteControlPort);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public FtpClient(String remoteHostName, int
remoteControlPort) throws IOException, FtpException{
    try{
      f=new FTPClient(remoteHostName, remoteControlPort);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void changeDirectory(String remoteDirectory) throws
IOException, FtpException{
    try{
      f.chdir(remoteDirectory);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public String[] dir() throws IOException, FtpException{
    try{
      return f.dir();
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public String[] dir(String remoteDirectory) throws
IOException, FtpException{
    try{
      return f.dir(remoteDirectory);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public FtpReply getLastValidReply(){
      return new FtpReply(f.getLastValidReply());
  }
```

```
  public void login(String username, String password) throws
IOException, FtpException{
    try{
      f.login(username,password);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void makeDirectory(String remoteDirectory) throws
IOException, FtpException{
    try{
      f.mkdir(remoteDirectory);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void removeDirectory(String remoteDirectory) throws
IOException, FtpException{
    try{
      f.rmdir(remoteDirectory);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void password (String password) throws IOException,
FtpException{
    try{
      f.password(password);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void put(String localPath, String remoteFile) throws
IOException, FtpException{
    try{
      f.put(localPath,remoteFile);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public String printWorkingDirectory() throws IOException,
FtpException{
    try{
      return f.pwd();
```

```
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void quit() throws IOException, FtpException{
    try{
      f.quit();
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void setConnectionMode (FtpConnectionMode mode){
    if (mode.connectionMode().equals("Active"))
      f.setConnectMode(FTPConnectMode.ACTIVE);
    else
      f.setConnectMode(FTPConnectMode.PASV);
  }

  public void setTransferType (FtpTransferType type) throws
IOException, FtpException{
    try{
      if (type.transferType().equals("Ascii"))
        f.setType(FTPTransferType.ASCII);
      else
        f.setType(FTPTransferType.BINARY);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public String remoteSystem() throws IOException,
FtpException{
    try{
      return f.system();
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }

  public void username(String username) throws IOException,
FtpException{
    try{
      f.user(username);
    }
    catch (FTPException e){
      throw new FtpException(e);
    }
  }
```

```
}
```

```
package ftp;

public abstract class FtpConnectionMode {

  public abstract String connectionMode();

}
```

```
package ftp;

import com.enterprisedt.net.ftp.*;

public class FtpException extends Exception{

  private int replyCode=0;
  FTPException e;

  public FtpException(FTPException e){
    this.e=e;
    replyCode=e.getReplyCode();
  }

  public int getReplyCode(){
    //Ritorna il codice di risposta del server, altrimenti -1
    return replyCode;
  }

  public String toString(){
    return e.toString();
  }

}
```

```
package ftp;

public class FtpPassiveConnectionMode extends
FtpConnectionMode {

  public String connectionMode(){
    return "Passive";
  }

}
```

```
package ftp;

import com.enterprisedt.net.ftp.*;

public class FtpReply {

  private String replyCode=null;
```

```
  private String replyText=null;

  public FtpReply(FTPReply r){
    replyCode=r.getReplyCode();
    replyText=r.getReplyText();
  }

  public String getReplyCode(){
    return replyCode;
  }

  public String getReplyText(){
    return replyText;
  }

}
                                  ---
package ftp;

public abstract class FtpTransferType {

  public abstract String transferType();

}
```

Package ftpmodel

```
package ftpmodel;

import java.net.*;

public class FtpAutentication{

  protected String username;
  protected String password;
  protected InetAddress server;

  public FtpAutentication(String username, String password,
String server) throws UnknownHostException{
    this.username=username;
    this.password=password;
    this.server=InetAddress.getByName(server);
  }

  public FtpAutentication(String username, String password,
InetAddress server){
    this.username=username;
    this.password=password;
    this.server=server;
  }

  public String getUsername(){
    return username;
  }

  public String getPassword(){
    return password;
  }

  public String getServerString(){
    return server.getHostAddress();
  }

  public InetAddress getServer(){
    return server;
  }

  public void setUsername(String username){
    this.username=username;
  }

  public void setPassword(String password){
    this.password=password;
  }

  public void setServer(InetAddress server){
    this.server=server;
  }
```

```
  public void setServerString(String server) throws
UnknownHostException{
    this.server=InetAddress.getByName(server);
  }

}
                              ---
package ftpmodel;

public class FtpHighPriority extends FtpPriority {

  public String priority(){
    return "HIGH";
  }

}
                              ---
package ftpmodel;

public class FtpLowPriority extends FtpPriority {

  public String priority(){
    return "LOW";
  }

}
                              ---
package ftpmodel;

public class FtpMediumPriority extends FtpPriority {

  public String priority(){
    return "MEDIUM";
  }

}
                              ---
package ftpmodel;

import java.util.*;
import java.io.*;

public class FtpMultipleTransfer extends FtpTransfer
implements IFtpMultipleTransfer{

  protected long identifier;
  protected FtpPriority priority;
  protected Date inQueueDateAndTime;

  public FtpMultipleTransfer() {
    super();
    this.priority= new FtpMediumPriority();
    this.inQueueDateAndTime=new Date();
    this.identifier=0;
```

```
  }

  public FtpMultipleTransfer(String localFileName, String
localPath, String remoteRelativePath, String remoteFileName){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me);
    this.priority= new FtpMediumPriority();
    this.inQueueDateAndTime=new Date();
    this.identifier=0;
  }

  public FtpMultipleTransfer(FtpPriority priority, Date
inQueueDateAndTime, long identifier){
    super();
    this.priority=priority;
    this.inQueueDateAndTime=inQueueDateAndTime;
    this.identifier=identifier;
  }

  public FtpMultipleTransfer(String localFileName, String
localPath, String remoteRelativePath, String remoteFileName,
FtpPriority priority, Date inQueueDateAndTime, long
identifier){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me);
    this.priority=priority;
    this.inQueueDateAndTime=inQueueDateAndTime;
    this.identifier=identifier;
  }


  public FtpPriority getPriority(){
    return priority;
  }

  public Date getInQueueDateAndTime(){
    return inQueueDateAndTime;
  }

  public long getIdentifier(){
    return identifier;
  }

  public void setPriority(FtpPriority priority){
      this.priority=priority;
  }

  public void setInQueueDateAndTime(Date inQueueDateAndTime){
    this.inQueueDateAndTime=inQueueDateAndTime;
  }

  public void setIdentifier(long identifier){
```

```
    this.identifier=identifier;
  }

}
                                ---
package ftpmodel;

import java.util.*;

public class FtpMultipleTransferDone extends
FtpMultipleTransfer implements IFtpMultipleTransferDone {

  protected Date startDateAndTime;
  protected Date stopDateAndTime;

  public FtpMultipleTransferDone(){
    super();
    this.startDateAndTime=new Date();
    this.stopDateAndTime=new Date();
  }

  public FtpMultipleTransferDone(Date startDateAndTime, Date
stopDateAndTime) {
    super();
    this.startDateAndTime=startDateAndTime;
    this.stopDateAndTime=stopDateAndTime;
  }

  public FtpMultipleTransferDone(String localFileName, String
localPath, String remoteRelativePath, String remoteFileName,
FtpPriority priority, Date inQueueDateAndTime, long
identifier){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier);
    this.startDateAndTime=new Date();
    this.stopDateAndTime=new Date();
  }

  public FtpMultipleTransferDone(String localFileName, String
localPath, String remoteRelativePath, String remoteFileName,
FtpPriority priority, Date inQueueDateAndTime, long
identifier, Date startDateAndTime, Date stopDateAndTime){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier);
    this.startDateAndTime=startDateAndTime;
    this.stopDateAndTime=stopDateAndTime;
  }

  public Date getStartDateAndTime(){
      return startDateAndTime;
  }
```

```
  public Date getStopDateAndTime(){
      return stopDateAndTime;
  }

  public void setStartDateAndTime(Date startDateAndTime){
    this.startDateAndTime=startDateAndTime;
  }

  public void setStopDateAndTime(Date stopDateAndTime){
    this.stopDateAndTime=stopDateAndTime;
  }

}
                              ---
package ftpmodel;

import java.util.*;

public class FtpMultipleTransferError extends
FtpMultipleTransfer implements IFtpMultipleTransferError {

  protected Date startDateAndTime;
  protected Date errorDateAndTime;

  public FtpMultipleTransferError(){
    super();
    this.startDateAndTime=new Date();
    this.errorDateAndTime=new Date();
  }

  public FtpMultipleTransferError(Date startDateAndTime, Date
errorDateAndTime) {
    super();
    this.startDateAndTime=startDateAndTime;
    this.errorDateAndTime=errorDateAndTime;
  }

  public FtpMultipleTransferError(String localFileName,
String localPath, String remoteRelativePath, String
remoteFileName, FtpPriority priority, Date
inQueueDateAndTime, long identifier){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier);
    this.startDateAndTime=new Date();
    this.errorDateAndTime=new Date();
  }

  public FtpMultipleTransferError(String localFileName,
String localPath, String remoteRelativePath, String
remoteFileName, FtpPriority priority, Date
inQueueDateAndTime, long identifier, Date startDateAndTime,
Date errorDateAndTime){
```

```
super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier);
    this.startDateAndTime=startDateAndTime;
    this.errorDateAndTime=errorDateAndTime;
  }

  public Date getStartDateAndTime(){
      return startDateAndTime;
  }

  public Date getErrorDateAndTime(){
      return errorDateAndTime;
  }

  public void setStartDateAndTime(Date startDateAndTime){
    this.startDateAndTime=startDateAndTime;
  }

  public void setErrorDateAndTime(Date errorDateAndTime){
    this.errorDateAndTime=errorDateAndTime;
  }

}
                              ---
package ftpmodel;

public abstract class FtpPriority {

  public abstract String priority();

  public static final FtpPriority parse(String priority){
    if (priority.equals("HIGH"))
      return new FtpHighPriority();
    else if (priority.equals("MEDIUM"))
      return new FtpMediumPriority();
    else if (priority.equals("LOW"))
      return new FtpLowPriority();
    else
      return new FtpMediumPriority();
  }

}

package ftpmodel;

public class FtpTransfer implements IFtpTransfer {

  protected String localFileName;
  protected String localPath;
  protected String remoteFileName;
  protected String remoteRelativePath;

  public FtpTransfer() {
```

```
    localFileName="";
    localPath="";
    remoteRelativePath="";
    remoteFileName="";
  }

  public FtpTransfer(String localFileName, String localPath,
String remoteRelativePath, String remoteFileName){
    this.localFileName=localFileName;
    this.localPath=localPath;
    this.remoteRelativePath=remoteRelativePath;
    this.remoteFileName=remoteFileName;
  }

  public String getLocalFileName(){
      return localFileName;
  }

  public String getLocalPath(){
      return localPath;
  }

  public String getRemoteRelativePath(){
      return remoteRelativePath;
  }

  public String getRemoteFileName(){
    return remoteFileName;
  }

  public void setLocalFileName(String localFileName){
      this.localFileName=localFileName;
  }

  public void setLocalPath(String localPath){
      this.localPath=localPath;
  }

  public void setRemoteRelativePath(String
remoteRelativePath){
      this.remoteRelativePath=remoteRelativePath;
  }

  public void setRemoteFileName(String remoteFileName){
      this.remoteFileName=remoteFileName;
  }

}
                              ---
package ftpmodel;

import java.util.*;

public interface IFtpMultipleTransfer extends IFtpTransfer {
```

```
  public FtpPriority getPriority();
  public Date getInQueueDateAndTime();
  public long getIdentifier();

  public void setPriority(FtpPriority priority);
  public void setInQueueDateAndTime(Date inQueueDateAndTime);
  public void setIdentifier(long identifier);

}
```

Package guicommons

```
package guicommons;

import java.awt.event.*;

public class ExitListener implements ActionListener {

  public ExitListener() {

  }

  public void actionPerformed(ActionEvent actionEvent) {
    System.exit(0);
  }

}
                                ---
package guicommons;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import tucsonagents.*;
import alice.logictuple.*;
import alice.tucson.api.*;

public class TupleCenterPanel extends JPanel {

  private TupleCentreId tupleCentre=null;
  private LogicTuple logicTupleTemplate=null;

  private JScrollPane pannelloScorrimento = new
JScrollPane();
  private JButton btAggiorna = new JButton();
  private JTextArea areaTuple = new JTextArea();
  private GridBagLayout gridBagLayoutMain = new
GridBagLayout();


  public TupleCenterPanel(TupleCentreId tupleCentre,
LogicTuple logicTupleTemplate) {

    this.tupleCentre=tupleCentre;
    this.logicTupleTemplate=logicTupleTemplate;

    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  private void jbInit() throws Exception {
```

```
    this.setLayout(gridBagLayoutMain);
    pannelloScorrimento.setAutoscrolls(true);
    btAggiorna.setForeground(Color.red);
    btAggiorna.setText("Update");
    btAggiorna.addActionListener(new
UpdateListener(tupleCentre,logicTupleTemplate,areaTuple));
    areaTuple.setMargin(new Insets(10, 10, 10, 10));
    areaTuple.setEditable(false);
    areaTuple.setToolTipText("");
    areaTuple.setBorder(null);
    areaTuple.setFont(new java.awt.Font("Dialog", 1, 12));
    this.add(pannelloScorrimento,  new GridBagConstraints(0,
0, 1, 1, 1.0, 1.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(9, 11, 0, 10), 293,
122));
    this.add(btAggiorna,  new GridBagConstraints(0, 1, 1, 1,
0.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(9, 11, 10, 10), 236, 0));
    pannelloScorrimento.getViewport().add(areaTuple, null);


  }

}

class UpdateListener implements ActionListener{

  TupleCentreId tupleCentre=null;
  LogicTuple logicTupleTemplate=null;
  JTextArea areaTuple=null;

  public UpdateListener(TupleCentreId tupleCentre, LogicTuple
logicTupleTemplate, JTextArea areaTuple){
    this.tupleCentre=tupleCentre;
    this.logicTupleTemplate=logicTupleTemplate;
    this.areaTuple=areaTuple;
  }

  public void actionPerformed(ActionEvent e){


    String listaTuple[];
    areaTuple.setText("");

    try{

listaTuple=RdAllAgent.rdAll(tupleCentre,logicTupleTemplate);
      for (int i=0; i<listaTuple.length; i++){
        listaTuple[i]=listaTuple[i]+"\n";
        areaTuple.append(listaTuple[i]);
      }
    }
```

```
    catch (UnreachableNodeException ex){
      System.out.println(ex.toString());
      JOptionPane.showMessageDialog(new JFrame(),"Cannot find
Tucson node.","Error",JOptionPane.ERROR_MESSAGE);
    }


  }

}
                              ---
package guicommons;

import java.awt.event.*;

public class WindowEventsListener implements WindowListener {
          public void windowClosed(WindowEvent e){}
          public void windowClosing(WindowEvent
e){System.exit(0);}
          public void windowOpened(WindowEvent e){}
          public void windowIconified(WindowEvent e){}
          public void windowDeiconified(WindowEvent e){}
          public void windowActivated(WindowEvent e){}
          public void windowDeactivated(WindowEvent e){}
}
```

Package test

```
package test;

import javax.swing.*;
import guicommons.*;

public class FtpInsertUploadAgentGui{

  public static void main(String[] args) {

    JFrame frame=new JFrame("Ftp upload test");
    frame.getContentPane().add(new
IUAAutenticationPanel(frame));
    frame.pack();
    frame.addWindowListener(new WindowEventsListener());
    frame.show();

  }

}
```

```
package test;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import ftp.*;
import ftpmodel.*;
import guicommons.*;
import tucsonagents.*;
import alice.tucson.api.*;
import tucsonftpmodel.*;

public class FtpUploadAgentCL {
  public static void main(String[] args) {

    /* Gli argomenti con cui lanciare l'applicazione sono:

    -server
    -username
    -password
    -porta

    */
    System.out.println("*** FtpUploadAgentCL ***");

    InputStreamReader isr=new InputStreamReader(System.in);

    if(args.length!=5){
      System.out.println("Usage: java FtpUploadAgentCL");
      System.out.println("<server name or address>");
```

```
      System.out.println("<username>");
      System.out.println("<password>");
      System.out.println("<server port (21 standard)>");
      System.out.println("<yes|no> (Use priority)");

      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
      System.exit(1);
    }

    String server=args[0], username=args[1],
password=args[2], port=args[3], priority=args[4];
    boolean priorita=false;
    int porta=21;
    FtpAutentication autenticazione=null;

    try{
      autenticazione=new
FtpAutentication(username,password,server);
    }
    catch (UnknownHostException e){
      System.out.println("Can't resolve host name.");
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
      System.exit(1);
    }
    try{
      porta=Integer.parseInt(port);
    }
    catch (NumberFormatException e){
      System.out.println("Port number not valid.");
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
      System.exit(1);
    }
    if (priority.toLowerCase().equals("yes"))
      priorita=true;
    else
      if (priority.toLowerCase().equals("no"))
```

```
        priorita=false;
    else{
      System.out.println("Priority use not valid.");
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception e){
        System.out.println(e.toString());
      }
      System.exit(1);
    }
    try{
      AgentId aid=new
AgentId(FtpUploadAgent.getDefaultAgentName());
      FtpUploadAgent agente=new
FtpUploadAgent(aid,autenticazione,priorita,porta,new
TucsonFtpMultipleTransfer(),new
TucsonFtpMultipleTransferDone(),new
TucsonFtpMultipleTransferError());
      agente.spawn();
      System.out.println("Agent started...");
    }
    catch (TucsonException e){
      System.out.println(e.toString());
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
      System.exit(1);
    }
    catch (IOException e){
      System.out.println(e.toString());
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
      System.exit(1);
    }
    catch (FtpException e){
      System.out.println(e.toString());
      System.out.println("Press enter key to continue...");
      try{
        isr.read();
      }
      catch(Exception ex){
        System.out.println(ex.toString());
      }
```

```
      System.exit(1);
    }
  }
}
                                ---
package test;

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.*;
import ftpmodel.*;
import guicommons.*;

public class IUAAutenticationPanel extends JPanel {

  JFrame frame=null;

  private JButton btExit = new JButton();
  private JButton btOk = new JButton();
  private JTextField txtServer = new JTextField();
  private JPasswordField txtPassword = new JPasswordField();
  private JLabel lblUsername = new JLabel();
  private JLabel lblServer = new JLabel();
  private JPanel pannelloAutenticazione = new JPanel();
  private JLabel lblPassword = new JLabel();
  private JTextField txtUsername = new JTextField();
  private GridBagLayout gridBagLayoutAute = new
GridBagLayout();
  private GridBagLayout gridBagLayoutPannello = new
GridBagLayout();

  public IUAAutenticationPanel(JFrame frame) {
    //Mantengo un riferimento al frame
    this.frame=frame;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setLayout(gridBagLayoutPannello);
    btExit.setText("Exit");
    btOk.setText("OK");
    lblUsername.setText("Username:");
    lblServer.setText("Server:");

pannelloAutenticazione.setBorder(BorderFactory.createEtchedBo
rder());
    pannelloAutenticazione.setLayout(gridBagLayoutAute);
    lblPassword.setText("Password:");
```

```
    pannelloAutenticazione.add(lblUsername,  new
GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(9, 7, 0, 0), 9, 0));
    pannelloAutenticazione.add(lblPassword,  new
GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(16, 7, 0, 0), 12, 0));
    pannelloAutenticazione.add(lblServer,  new
GridBagConstraints(0, 2, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(14, 7, 8, 0), 33, 4));
    pannelloAutenticazione.add(txtServer,  new
GridBagConstraints(1, 2, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(14, 11, 8, 11),
169, 0));
    pannelloAutenticazione.add(txtUsername,  new
GridBagConstraints(1, 0, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(9, 11, 0, 85), 95,
0));
    pannelloAutenticazione.add(txtPassword,  new
GridBagConstraints(1, 1, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(13, 11, 0, 85), 95,
0));
    this.add(btOk,  new GridBagConstraints(1, 1, 1, 1, 0.0,
0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(10, 111, 14, 13), 32,
7));
    this.add(btExit,  new GridBagConstraints(0, 1, 1, 1, 0.0,
0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(10, 8, 14, 0), 30, 7));
    this.add(pannelloAutenticazione,  new
GridBagConstraints(0, 0, 2, 1, 1.0, 1.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(7, 8, 0, 13), 0, -3));

    btExit.addActionListener(new ExitListener());
    btOk.addActionListener(new
OkListener(txtUsername,txtServer,txtPassword,this,frame));
  }

}


//Insieme dei listener per AutenticationPanel

class OkListener implements ActionListener {

  private JTextField username=null;
```

```
  private JTextField server=null;
  private JPasswordField password=null;
  private JPanel panel=null;
  private JFrame frame=null;

  public OkListener(JTextField username, JTextField server,
JPasswordField password, JPanel panel, JFrame frame) {
    this.username=username;
    this.server=server;
    this.password=password;
    this.panel=panel;
    this.frame=frame;
  }

  public void actionPerformed(ActionEvent actionEvent) {
    try{
      FtpAutentication autentication=new
FtpAutentication(username.getText(),String.valueOf(password.g
etPassword()),server.getText());
      frame.getContentPane().remove(panel);
      frame.getContentPane().add(new
IUATransferPanel(frame,autentication));
      frame.pack();
    }
    catch(UnknownHostException e){
      JOptionPane.showMessageDialog(frame,"Cannot resolve
host name.","Error",JOptionPane.ERROR_MESSAGE);
    }
  }

}
                                ---
package test;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import ftpmodel.*;
import guicommons.*;
import tucsonagents.*;
import tucsonftpmodel.*;
import alice.tucson.api.*;

public class IUATransferPanel extends JPanel {

  private FtpInsertUploadAgent insert=null;
  private FtpAutentication autentication=null;
  private JFrame frame=null;
  private JFrame frameUpload=null, frameError=null,
frameDone=null;

  //Grafica
  private JButton btInsert = new JButton();
  private JButton btDone = new JButton();
```

```
  private JComboBox cmbPriority = new JComboBox();
  private JPanel pannelloInserimento = new JPanel();
  private JLabel lblRemotePath = new JLabel();
  private JLabel lblRemoteFilename = new JLabel();
  private JLabel lblLocalFilename = new JLabel();
  private JTextField txtRemotePath = new JTextField();
  private JTextField txtLocalPath = new JTextField();
  private JTextField txtRemoteFilename = new JTextField();
  private JTextField txtLocalFilename = new JTextField();
  private JLabel lblLocalPath = new JLabel();
  private JLabel lblPriority = new JLabel();
  private JButton btChoose = new JButton();
  private JButton btShow = new JButton();
  private GridBagLayout gridBagLayoutCampi = new
GridBagLayout();
  private GridBagLayout gridBagLayoutTutto = new
GridBagLayout();


  public IUATransferPanel(JFrame frame, FtpAutentication
autentication) {

    //Mantengo un riferimento al frame originario
    this.frame=frame;
    this.autentication=autentication;

    //Creazione dei tre pannelli per vedere i centri di tuple
    frameUpload=new JFrame("Queued tranfers");
    frameError=new JFrame("Incorrect transfers");
    frameDone=new JFrame("Done transfers");
    try{
      frameUpload.getContentPane().add(new
TupleCenterPanel(new
TupleCentreId(FtpUploadAgent.getUploadTupleCentreName(autenti
cation)),new
TucsonFtpMultipleTransfer().getLogicTupleTemplate()));
      frameError.getContentPane().add(new
TupleCenterPanel(new
TupleCentreId(FtpUploadAgent.getErrorTupleCentreName(autentic
ation)),new
TucsonFtpMultipleTransferError().getLogicTupleTemplate()));
      frameDone.getContentPane().add(new TupleCenterPanel(new
TupleCentreId(FtpUploadAgent.getDoneTupleCentreName(autentica
tion)),new
TucsonFtpMultipleTransferDone().getLogicTupleTemplate()));
    }
    catch (TucsonException e){
      System.out.println(e.toString());
      System.exit(1);
    }
    frameUpload.pack();
    frameError.pack();
    frameDone.pack();
```

```
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setToolTipText("");
    this.setLayout(gridBagLayoutTutto);
    btInsert.setFont(new java.awt.Font("Dialog", 1, 14));
    btInsert.setForeground(Color.red);
    btInsert.setText("Insert");
    btDone.setText("Done");
    cmbPriority.setAutoscrolls(true);
    cmbPriority.setToolTipText("");
    cmbPriority.setMaximumRowCount(3);
    pannelloInserimento.setLayout(gridBagLayoutCampi);
    lblRemotePath.setText("Remote path:");
    lblRemoteFilename.setText("Remote file name:");
    lblLocalFilename.setText("Local file name:");
    lblLocalPath.setText("Local path:");
    lblPriority.setText("Priority:");
    btChoose.setText("Choose...");
    btShow.setText("Show tuple centres");
    this.add(pannelloInserimento,  new GridBagConstraints(0,
0, 3, 1, 1.0, 1.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(14, 12, 0, 11), 0, -3));
    pannelloInserimento.add(lblPriority,  new
GridBagConstraints(0, 5, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(24, 24, 21, 14), 49, 4));
    pannelloInserimento.add(cmbPriority,  new
GridBagConstraints(1, 5, 1, 1, 1.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.HORIZONTAL, new Insets(24, 7, 21, 99), 19,
0));
    pannelloInserimento.add(txtRemoteFilename,  new
GridBagConstraints(1, 4, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(8, 7, 0, 155), 84,
-1));
    pannelloInserimento.add(lblRemoteFilename,  new
GridBagConstraints(0, 4, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(8, 24, 0, 0), 2, 4));
    pannelloInserimento.add(lblLocalPath,  new
GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(22, 24, 0, 25), 18, 9));
    pannelloInserimento.add(txtLocalPath,  new
GridBagConstraints(1, 0, 1, 1, 1.0, 0.0
```

```
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(22, 7, 0, 30), 209,
-1));
    pannelloInserimento.add(txtLocalFilename,  new
GridBagConstraints(1, 1, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(0, 7, 0, 155), 84,
-1));
    pannelloInserimento.add(lblLocalFilename,  new
GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(0, 24, 0, 14), 2, 4));

    pannelloInserimento.add(txtRemotePath,  new
GridBagConstraints(1, 3, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(20, 7, 0, 30), 209,
-1));
    pannelloInserimento.add(lblRemotePath,  new
GridBagConstraints(0, 3, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(20, 24, 0, 20), 9, 4));
    pannelloInserimento.add(btChoose,  new
GridBagConstraints(1, 2, 1, 1, 0.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(8, 7, 0, 30), 125, -7));
    this.add(btShow,  new GridBagConstraints(0, 1, 1, 1, 0.0,
0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(14, 12, 8, 0), 7, 7));
    this.add(btDone,  new GridBagConstraints(2, 1, 1, 1, 0.0,
0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(14, 8, 8, 11), 2, 7));
    this.add(btInsert,  new GridBagConstraints(1, 1, 1, 1,
0.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(14, 9, 8, 0), 72, 3));


    //Impostazione elementi del combo
    String[] elementiCombo=new String[3];
    elementiCombo[0]="LOW";
    elementiCombo[1]="MEDIUM";
    elementiCombo[2]="HIGH";
    DefaultComboBoxModel model=new
DefaultComboBoxModel(elementiCombo);
    cmbPriority.setModel(model);

    //Associazione dei listener ai pulsanti
    btDone.addActionListener(new DoneListener(frame,this,
frameUpload, frameError,frameDone));
```

```
    btInsert.addActionListener(new
InsertListener(txtLocalPath,txtLocalFilename,txtRemotePath,tx
tRemoteFilename,cmbPriority,autentication,frame));
    btChoose.addActionListener(new
ChooseListener(txtLocalPath,txtLocalFilename,frame));
    btShow.addActionListener(new ShowListener(frameUpload,
frameError,frameDone));

  }
}


class DoneListener implements ActionListener{

  JFrame frame=null;
  JPanel panel=null;
  JFrame frameUpload=null, frameError=null,frameDone=null;

  public DoneListener(JFrame frame, JPanel panel, JFrame
frameUpload, JFrame frameError, JFrame frameDone){
    this.frame=frame;
    this.panel=panel;
    this.frameUpload=frameUpload;
    this.frameError=frameError;
    this.frameDone=frameDone;
  }

  public void actionPerformed(ActionEvent e){
    frameUpload.dispose();
    frameError.dispose();
    frameDone.dispose();
    frame.getContentPane().remove(panel);
    frame.getContentPane().add(new
IUAAutenticationPanel(frame));
    frame.pack();
  }

}


class InsertListener implements ActionListener{

  JTextField localPath=null,localFilename=null,
remotePath=null, remoteFilename=null;
  JComboBox priority=null;
  FtpAutentication autentication=null;
  JFrame frame;

  public InsertListener(JTextField localPath, JTextField
localFilename, JTextField remotePath, JTextField
remoteFilename, JComboBox priority, FtpAutentication
autentication, JFrame frame){
    this.localPath=localPath;
    this.localFilename=localFilename;
```

```
    this.remotePath=remotePath;
    this.remoteFilename=remoteFilename;
    this.priority=priority;
    this.frame=frame;
    this.autentication=autentication;
  }

  public void actionPerformed(ActionEvent e){

    FtpInsertUploadAgent insert=new
FtpInsertUploadAgent(autentication,new
TucsonFtpMultipleTransfer());

    //Imposto il contenuto di insert con quello dei campi
    insert.getTransfer().setLocalPath(localPath.getText());

insert.getTransfer().setLocalFileName(localFilename.getText()
);

insert.getTransfer().setRemoteRelativePath(remotePath.getText
());

insert.getTransfer().setRemoteFileName(remoteFilename.getText
());

insert.getTransfer().setPriority(FtpPriority.parse(priority.g
etSelectedItem().toString()));

    //Parte di inserimento in Tucson
    try{
      insert.toTucson();
    }
    catch (UnreachableNodeException exception){
      JOptionPane.showMessageDialog(frame,"Cannot find Tucson
node.","Error",JOptionPane.ERROR_MESSAGE);
    }

  }

}


class ChooseListener implements ActionListener{

  JTextField path, file=null;
  JFrame frame;

  public ChooseListener(JTextField path, JTextField file,
JFrame frame){
    this.path=path;
    this.file=file;
    this.frame=frame;
  }
```

```
  public void actionPerformed(ActionEvent e){
    JFileChooser d=new JFileChooser();
    //d.setCurrentDirectory(new File("."));
    int result=d.showOpenDialog(frame);
    if (result==JFileChooser.APPROVE_OPTION){
      file.setText(d.getSelectedFile().getName());
      path.setText(d.getSelectedFile().getParent());
    }
  }

}

class ShowListener implements ActionListener{

  JFrame upload=null,error=null,done=null;

  public ShowListener(JFrame upload,JFrame error,JFrame
done){
    this.upload=upload;
    this.error=error;
    this.done=done;
  }

  public void actionPerformed(ActionEvent e){
    upload.show();
    error.show();
    done.show();
  }

}
                              ---
package test;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import javax.swing.*;
import ftp.*;
import ftpmodel.*;
import guicommons.*;
import tucsonagents.*;
import alice.tucson.api.*;
import tucsonftpmodel.*;

public class UAMainPanel extends JPanel {

  JFrame frame=null;

  private JButton btExit = new JButton();
  private JTextField txtServer = new JTextField();
  private JPasswordField txtPassword = new JPasswordField();
  private JLabel lblUsername = new JLabel();
  private JLabel lblServer = new JLabel();
```

```
  private JPanel pannelloAutenticazione = new JPanel();
  private JLabel lblPassword = new JLabel();
  private JTextField txtUsername = new JTextField();
  private JCheckBox chkUsePriority = new JCheckBox();
  private JButton btStart = new JButton();
  private JTextField txtPort = new JTextField();
  private JLabel lblPort = new JLabel();
  private GridBagLayout gridBagLayoutPannello = new
GridBagLayout();
  private GridBagLayout gridBagLayoutMainPannello = new
GridBagLayout();

  public UAMainPanel(JFrame frame) {
    this.frame=frame;
    //Mantengo un riferimento al frame
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setLayout(gridBagLayoutMainPannello);
    btExit.setText("Exit");
    lblUsername.setText("Username:");
    lblServer.setText("Server:");

pannelloAutenticazione.setBorder(BorderFactory.createEtchedBo
rder());
    pannelloAutenticazione.setLayout(gridBagLayoutPannello);
    lblPassword.setText("Password:");
    chkUsePriority.setText("Use priority");
    btStart.setForeground(Color.blue);
    btStart.setText("Start");
    txtPort.setText("21");
    txtPort.setHorizontalAlignment(SwingConstants.RIGHT);
    lblPort.setText("Server port:");
    pannelloAutenticazione.add(txtUsername,  new
GridBagConstraints(1, 0, 2, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(20, 11, 0, 15),
198, 0));
    pannelloAutenticazione.add(lblUsername,  new
GridBagConstraints(0, 0, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(20, 9, 0, 0), 9, 0));
    pannelloAutenticazione.add(lblPassword,  new
GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(16, 9, 0, 0), 12, 0));
    pannelloAutenticazione.add(txtPassword,  new
GridBagConstraints(1, 1, 2, 1, 1.0, 0.0
```

```
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(13, 11, 0, 15),
198, 0));
    pannelloAutenticazione.add(lblServer,  new
GridBagConstraints(0, 2, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(16, 9, 0, 0), 33, 4));
    pannelloAutenticazione.add(txtServer,  new
GridBagConstraints(1, 2, 2, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(16, 11, 0, 15),
198, 0));
    pannelloAutenticazione.add(txtPort,  new
GridBagConstraints(1, 3, 1, 1, 1.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.HORIZONTAL, new Insets(12, 11, 0, 60), 27,
0));
    pannelloAutenticazione.add(chkUsePriority,  new
GridBagConstraints(1, 4, 1, 1, 0.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(8, 11, 12, 0), 18, 1));
    this.add(pannelloAutenticazione,  new
GridBagConstraints(0, 0, 1, 1, 1.0, 1.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.BOTH, new Insets(7, 8, 0, 12), 0, -1));
    this.add(btExit,  new GridBagConstraints(0, 1, 1, 1, 0.0,
0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(10, 92, 10, 91), 96, 7));
    pannelloAutenticazione.add(lblPort,  new
GridBagConstraints(0, 3, 1, 1, 0.0, 0.0
            ,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(12, 9, 0, 0), 9, 4));
    pannelloAutenticazione.add(btStart,  new
GridBagConstraints(2, 4, 1, 1, 0.0, 0.0
            ,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(0, 37, 12, 15), 1, 7));
    btStart.addActionListener(new
StartListener(txtUsername,txtServer,txtPassword,txtPort,chkUs
ePriority,btStart,frame));
    btExit.addActionListener(new ExitListener());
  }


}


//Insieme dei listener per AutenticationPanel

class StartListener implements ActionListener {

  private JTextField username=null;
  private JTextField server=null;
  private JPasswordField password=null;
```

```
  private JTextField port=null;
  JCheckBox usePriority=null;
  private JButton startButton=null;
  private JFrame frame;

  public StartListener(JTextField username, JTextField
server, JPasswordField password, JTextField port, JCheckBox
usePriority, JButton startButton,JFrame frame) {
    this.username=username;
    this.server=server;
    this.password=password;
    this.port=port;
    this.usePriority=usePriority;
    this.startButton=startButton;
    this.frame=frame;
  }

  public void actionPerformed(ActionEvent actionEvent) {

    try{
      boolean priorita=usePriority.isSelected();
      int porta;
      try{
        porta=Integer.parseInt(port.getText());
      }
      catch (NumberFormatException e){
        porta=21;
        port.setText("21");
      }
      FtpAutentication autenticazione=new
FtpAutentication(username.getText(),String.valueOf(password.g
etPassword()),server.getText());
      AgentId aid=new
AgentId(FtpUploadAgent.getDefaultAgentName());
      FtpUploadAgent agente=new
FtpUploadAgent(aid,autenticazione,priorita,porta,new
TucsonFtpMultipleTransfer(),new
TucsonFtpMultipleTransferDone(),new
TucsonFtpMultipleTransferError());

      agente.spawn();
      startButton.setEnabled(false);
      port.setEnabled(false);
      usePriority.setEnabled(false);
      server.setEnabled(false);
      username.setEnabled(false);
      password.setEnabled(false);
    }
    catch (UnknownHostException e){
      System.out.println(e.toString());
      JOptionPane.showMessageDialog(frame,"Cannot resolve
host name.","Error",JOptionPane.ERROR_MESSAGE);
    }
    catch (TucsonException e){
```

```
      //Errore critico
      System.out.println(e.toString());
      System.exit(1);
    }
    catch (IOException e){
      System.out.println(e.toString());
      JOptionPane.showMessageDialog(frame,"Cannot establish
FTP connection.","Error",JOptionPane.ERROR_MESSAGE);
    }
    catch (FtpException e){
      System.out.println(e.toString());
      JOptionPane.showMessageDialog(frame,"Cannot establish
FTP connection.","Error",JOptionPane.ERROR_MESSAGE);
    }



  }

}
```

Package tucsonagents

```
package tucsonagents;

import java.util.*;
import ftpmodel.*;
import tucsonftpmodel.*;
import alice.logictuple.*;
import alice.tucson.api.*;

public class FtpInsertUploadAgent {

  private FtpAutentication autentication=null;
  private ITucsonFtpMultipleTransfer transfer=null;
  private TucsonContext cn=null;
  private TupleCentreId tid=null;

  public final static String getDefaultAgentName(){
    return "ftpInsertUploadAgent";
  }

  public FtpInsertUploadAgent(FtpAutentication autentication,
ITucsonFtpMultipleTransfer transfer){
    this.autentication=autentication;
    this.transfer=transfer;
  }

  public FtpAutentication getAutentication(){
    return autentication;
  }

  public void setFtpAutentication(FtpAutentication
autentication){
    this.autentication=autentication;
  }

  public ITucsonFtpMultipleTransfer getTransfer(){
    return transfer;
  }

  public void setTransfer(ITucsonFtpMultipleTransfer
transfer){
    this.transfer=transfer;
  }


  public void toTucson() throws UnreachableNodeException{

    transfer.setInQueueDateAndTime(new Date());
    LogicTuple tuplaTucson=transfer.getLogicTuple();

    try{
```

```
      cn=Tucson.enterContext(new
DefaultContextDescription(new
AgentId(getDefaultAgentName())));
      tid=new
TupleCentreId(FtpUploadAgent.getUploadTupleCentreName(autenti
cation));
      cn.out(tid,tuplaTucson);
    }
    catch (UnreachableNodeException e){
      System.out.println(e.toString());
      throw new UnreachableNodeException();
    }
    catch(Exception e){
      System.out.println(e.toString());
      System.exit(1);
    }

  }

}
                              ---
package tucsonagents;

import java.io.*;
import java.text.*;
import java.util.*;
import ftp.*;
import ftpmodel.*;
import tucsonftpmodel.*;
import alice.logictuple.*;
import alice.tucson.api.*;

public class FtpUploadAgent extends Agent{

  protected FtpAutentication autentication=null;
  protected ITucsonFtpMultipleTransfer ftpTransfer=null;
  protected ITucsonFtpMultipleTransferError
ftpTransferError=null;
  protected  ITucsonFtpMultipleTransferDone
ftpTransferDone=null;
  protected  TupleCentreId tidUpload=null;
  protected  TupleCentreId tidDone=null;
  protected  TupleCentreId tidError=null;
  protected  boolean usePriority;
  protected  int remoteControlPort;
  protected  FtpClient clientFtp=null;


  public final static String getDefaultAgentName(){
    return "ftpUploadAgent";
  }

  public static String
getUploadTupleCentreName(FtpAutentication autentication){
```

```
    String username=autentication.getUsername();
    username=username.replace('.','_');
    String server=autentication.getServerString();
    server=server.replace('.','_');
    String
nomeCentroDiTuple="u"+username+"s"+server+"Upload";
    return nomeCentroDiTuple;
  }

  public static String
getErrorTupleCentreName(FtpAutentication autentication){
    String username=autentication.getUsername();
    username=username.replace('.','_');
    String server=autentication.getServerString();
    server=server.replace('.','_');
    String nomeCentroDiTuple="u"+username+"s"+server+"Error";
    return nomeCentroDiTuple;
  }

  public static String
getDoneTupleCentreName(FtpAutentication autentication){
    String username=autentication.getUsername();
    username=username.replace('.','_');
    String server=autentication.getServerString();
    server=server.replace('.','_');
    String nomeCentroDiTuple="u"+username+"s"+server+"Done";
    return nomeCentroDiTuple;
  }

  public FtpUploadAgent(AgentId aid, FtpAutentication
autentication, boolean usePriority, int remoteControlPort,
ITucsonFtpMultipleTransfer ftpTransfer,
ITucsonFtpMultipleTransferDone ftpTransferDone,
ITucsonFtpMultipleTransferError ftpTransferError) throws
TucsonException, FtpException, IOException{

    super(aid);
    this.autentication=autentication;
    this.ftpTransfer=ftpTransfer;
    this.ftpTransferDone=ftpTransferDone;
    this.ftpTransferError=ftpTransferError;
    this.usePriority=usePriority;
    this.remoteControlPort=remoteControlPort;

    //Inizializzazione degli id dei centri di tuple
    try{
      tidUpload=new
TupleCentreId(FtpUploadAgent.getUploadTupleCentreName(autenti
cation));
      tidDone=new
TupleCentreId(FtpUploadAgent.getDoneTupleCentreName(autentica
tion));
```

```
      tidError=new
TupleCentreId(FtpUploadAgent.getErrorTupleCentreName(autentic
ation));
    }
    catch (InvalidTupleCentreIdException e){
      //Errore critico.
      System.out.println(e.toString());
      System.exit(1);
    }

    //Connessione al server
    System.out.println("Connecting to server
"+autentication.getServerString()+"...");
    clientFtp=new
FtpClient(autentication.getServer(),remoteControlPort);
    //Autenticazione al server
    clientFtp.username(autentication.getUsername());
    stampaUltimaRisposta();
    clientFtp.password(autentication.getPassword());
    stampaUltimaRisposta();
    clientFtp.setConnectionMode(new
FtpActiveConnectionMode());
    clientFtp.setTransferType(new FtpBinaryTransferType());
    stampaUltimaRisposta();

  }

  protected void body(){

    while(true){

      try{

        LogicTuple tuplaLetta=null;
        if (usePriority==true){
          //Utilizzo la priorità

tuplaLetta=rdp(tidUpload,ftpTransfer.getLogicTupleTemplate(ne
w FtpHighPriority()));
          if (tuplaLetta==null){

tuplaLetta=rdp(tidUpload,ftpTransfer.getLogicTupleTemplate(ne
w FtpMediumPriority()));
            if (tuplaLetta==null)

tuplaLetta=rdp(tidUpload,ftpTransfer.getLogicTupleTemplate(ne
w FtpLowPriority()));
            //Se lo spazio di tuple è vuoto mi blocco
            if (tuplaLetta==null){

tuplaLetta=rd(tidUpload,ftpTransfer.getLogicTupleTemplate());
            }
          }
        }
```

```
        else
          //Non utilizzo la priorità

tuplaLetta=rdp(tidUpload,ftpTransfer.getLogicTupleTemplate())
;
        //Se lo spazio di tuple è vuoto mi blocco
        if (tuplaLetta==null){
          //Chiudo la connessione col client
          try{
            System.out.println("No more transfers found.
Closing FTP connection...");
            clientFtp.quit();
            stampaUltimaRisposta();
          }
          catch (IOException e){
            System.out.println(e.toString());
            System.exit(1);
          }
          catch (FtpException e){
            System.out.println(e.toString());
            System.exit(1);
          }

tuplaLetta=rd(tidUpload,ftpTransfer.getLogicTupleTemplate());
          try{
            System.out.println("Connecting to server
"+autentication.getServerString()+"...");
            //Connessione al server
            clientFtp=new
FtpClient(autentication.getServer(),remoteControlPort);
            //Autenticazione al server
            clientFtp.username(autentication.getUsername());
            stampaUltimaRisposta();
            clientFtp.password(autentication.getPassword());
            stampaUltimaRisposta();
            clientFtp.setConnectionMode(new
FtpActiveConnectionMode());
            clientFtp.setTransferType(new
FtpBinaryTransferType());
            stampaUltimaRisposta();
          }
          catch (IOException e){
            System.out.println(e.toString());
            System.exit(1);
          }
          catch (FtpException e){
            System.out.println(e.toString());
            System.exit(1);
          }
        }

        //Ho una tupla valida: pronto per il trasferimento
        try{
          ftpTransfer.parse(tuplaLetta);
```

```
          //Se riesco ad effettuare il parsing della tupla
procedo con il trasferimento
          trasferimento();
        }
        catch(InvalidTupleOperationException e){
          //Errore critico.
          System.out.println(e.toString());
          System.exit(1);
        }
        catch(ParseException e){
          //Errore critico.
          System.out.println(e.toString());
          System.exit(1);
        }
      }
      catch (InvalidLogicTupleException e){
        //Errore critico.
        System.out.println(e.toString());
        System.exit(1);
      }
      catch (TucsonException e){
        //Errore critico.
        System.out.println(e.toString());
        System.exit(1);
      }
      //**********************
    }

  }



  private void trasferimento(){

    Date dataEOraDiInizio=new Date();

    //Imposto tutti i campi dell'errore di trasferimento

ftpTransferError.setIdentifier(ftpTransfer.getIdentifier());

ftpTransferError.setLocalFileName(ftpTransfer.getLocalFileNam
e());

ftpTransferError.setLocalPath(ftpTransfer.getLocalPath());
    ftpTransferError.setPriority(ftpTransfer.getPriority());

ftpTransferError.setRemoteFileName(ftpTransfer.getRemoteFileN
ame());

ftpTransferError.setRemoteRelativePath(ftpTransfer.getRemoteR
elativePath());
    ftpTransferError.setStartDateAndTime(dataEOraDiInizio);
```

```
ftpTransferError.setInQueueDateAndTime(ftpTransfer.getInQueue
DateAndTime());

    //Imposto tutti i campi del trasferimento effettuato
eccetto la data e ora di fine

ftpTransferDone.setIdentifier(ftpTransfer.getIdentifier());

ftpTransferDone.setLocalFileName(ftpTransfer.getLocalFileName
());
    ftpTransferDone.setLocalPath(ftpTransfer.getLocalPath());
    ftpTransferDone.setPriority(ftpTransfer.getPriority());

ftpTransferDone.setRemoteFileName(ftpTransfer.getRemoteFileNa
me());

ftpTransferDone.setRemoteRelativePath(ftpTransfer.getRemoteRe
lativePath());
    ftpTransferDone.setStartDateAndTime(dataEOraDiInizio);

ftpTransferDone.setInQueueDateAndTime(ftpTransfer.getInQueueD
ateAndTime());

    try{

      /* Trasferimento ftp */
      try{

        clientFtp.changeDirectory("/");
        stampaUltimaRisposta();

        if (!ftpTransfer.getRemoteRelativePath().equals("")){
          String
pathRemoto=ftpTransfer.getRemoteRelativePath();
          pathRemoto=pathRemoto.replace('\\','/');
          String[] alberoDirectory=pathRemoto.split("/");
          String[] contenutoDirettorioRemoto=null;
          for (int i=0; i<alberoDirectory.length; i++){
            //Esamino il contenuto del direttorio remoto per
vedere se c'è la directory
            contenutoDirettorioRemoto=clientFtp.dir();
            if (contenutoDirettorioRemoto.length==0){
              clientFtp.makeDirectory(alberoDirectory[i]);
              stampaUltimaRisposta();
            }
            else{
              for (int j=0;
j<contenutoDirettorioRemoto.length; j++){
                if
(contenutoDirettorioRemoto[j].equals(alberoDirectory[i])){
                  //Se trovo la directory esco dal ciclo
                  break;
                }
```

```
                else{
                  if (j==(contenutoDirettorioRemoto.length-
1)){
                    //Se non ho trovato la directory ed è
l'ultimo giro
                    //La creo

clientFtp.makeDirectory(alberoDirectory[i]);
                    stampaUltimaRisposta();
                  }
                }
              }
            }
            //Mi sposto sulla directory trovata o creata
            clientFtp.changeDirectory(alberoDirectory[i]);
            stampaUltimaRisposta();
          }
        }

        //Sono puntato sulla directory remota corretta
        System.out.println("Starting transfer of:
"+ftpTransfer.getLocalFileName()+" ->
"+ftpTransfer.getRemoteFileName());

        System.out.println("File transfer in act...");

clientFtp.put(ftpTransfer.getLocalPath()+"/"+ftpTransfer.getL
ocalFileName(),ftpTransfer.getRemoteFileName());
        stampaUltimaRisposta();
        operazioniSuccesso();

      }
      catch (IOException e){


        operazioniErrore();
        System.out.println(e.toString());

      }
      catch (FtpException e){


        operazioniErrore();
        System.out.println(e.toString());

      }
      /* Fine Trasferimento ftp*/

    }
    catch(InvalidLogicTupleException e){
      //Errore critico.
      System.out.println(e.toString());
      System.exit(1);
    }
```

```
    catch(TucsonException e){
      //Errore critico.
      System.out.println(e.toString());
      System.exit(1);
    }

  }

  private void operazioniErrore() throws
InvalidLogicTupleException,TucsonException{
    //Scrivo data e ora di errore
    Date dataEOraDiErrore=new Date();
    ftpTransferError.setErrorDateAndTime(dataEOraDiErrore);

    //Le due istruzioni seguenti dovrebbero essere atomiche
per non avere perdite
    //Tupla di errore inserita nel centro di tuple
    out(tidError,ftpTransferError.getLogicTuple());
    //Rimuovo la tupla dallo spazio di tuple di upload

System.out.println(ftpTransfer.getLogicTuple().toString());
    in(tidUpload,ftpTransfer.getLogicTuple());
  }

  private void operazioniSuccesso() throws
InvalidLogicTupleException,TucsonException{
    //Scrivo data e ora di fine
    Date dataEOraDiFine=new Date();
    ftpTransferDone.setStopDateAndTime(dataEOraDiFine);

    //Le due istruzioni seguenti dovrebbero essere atomiche
per non avere perdite
    //Rimuovo la tupla dallo spazio di tuple di upload
    in(tidUpload,ftpTransfer.getLogicTuple());
    //Preparo la tupla di trasferimento effettuato e la
inserisco nel centro di tuple
    out(tidDone,ftpTransferDone.getLogicTuple());
  }

  private void stampaUltimaRisposta(){
    if (clientFtp.getLastValidReply()!=null){

System.out.println(clientFtp.getLastValidReply().getReplyCode
()+ " "+clientFtp.getLastValidReply().getReplyText());
    }
  }

}
```

```
package tucsonagents;

import java.io.*;
import java.util.*;
import alice.logictuple.*;
```

```
import alice.logictuple.Var;
import alice.tucson.api.*;
import alice.tuprolog.*;

public class RdAllAgent{

  public final static String getDefaultAgentName(){
    return "rdAllAgent";
  }

  public static String[] rdAll(TupleCentreId tupleCentre,
LogicTuple logicTupleTemplate) throws
UnreachableNodeException{

    String arrayStringhe[]=null;

    try{

      //Carico lo script respect nel centro di tuple
      TucsonContext cn=Tucson.enterContext(new
DefaultContextDescription(new
AgentId(getDefaultAgentName())));
      InputStream is = new
FileInputStream("../../Script/rdall.rsp");
      BufferedInputStream in = new BufferedInputStream(is);
      String text = alice.util.Tools.loadText(in);
      cn.setSpec(tupleCentre,text);


      Var vt=new Var("Lista");
      LogicTuple tlist=cn.in(tupleCentre,new
LogicTuple("rdall",new
TupleArgument(logicTupleTemplate.toTerm()),vt));
      Term t=vt.toTerm();
      Struct lista=(Struct) t.getTerm();
      alice.tuprolog.Var head=null;

      Vector listaTuple=new Vector();

      while(!lista.unify(new Struct())){
        head=new alice.tuprolog.Var("Head");
        alice.tuprolog.Var tail=new
alice.tuprolog.Var("Tail");
        boolean res=lista.unify(new Struct(head,tail));
        //System.out.println("Head: "+head);
        //System.out.println("Tail: "+tail);
        lista=(Struct) tail.getTerm();
        String tupla=head.toString();

        System.out.println(tupla);
        LogicTuple tuple=null;
        try{
          tuple=LogicTuple.parse(tupla);
```

```
          //Estraggo la tupla e la inserisco come stringa nel
vector
          listaTuple.add(tuple.getArg(1).toString());
        }
        catch (InvalidLogicTupleException e){
          listaTuple.add(new String("Can't parse!"));
        }

      }

      //Creo l'array di stringhe da ritornare
      arrayStringhe=new String[listaTuple.size()];
      listaTuple.copyInto(arrayStringhe);

    }
    catch (UnreachableNodeException e){
      System.out.println(e.toString());
      throw new UnreachableNodeException();
    }
    catch (Exception e){
      System.out.println(e.toString());
      arrayStringhe=new String[0];
    }
    return arrayStringhe;

  }

}
```

Package tucsonftpmodel

```
package tucsonftpmodel;

import ftpmodel.*;
import tucsonmodel.*;

public interface ITucsonFtpMultipleTransfer extends
ITucsonConverter, IFtpMultipleTransfer
{
}
                              ---
package tucsonftpmodel;

import ftpmodel.*;
import tucsonmodel.*;

public interface ITucsonFtpMultipleTransferDone extends
ITucsonConverter, IFtpMultipleTransferDone{
}
                              ---
package tucsonftpmodel;

import ftpmodel.*;
import tucsonmodel.*;

public interface ITucsonFtpMultipleTransferError extends
ITucsonConverter, IFtpMultipleTransferError{
}
                              ---
package tucsonftpmodel;

import java.text.*;
import java.util.*;
import ftpmodel.*;
import alice.logictuple.*;

public class TucsonFtpMultipleTransfer extends
FtpMultipleTransfer implements ITucsonFtpMultipleTransfer{

  private final String nomeTuplaLogica="ftpUpload";

  public TucsonFtpMultipleTransfer(){
    super();
  }

  public TucsonFtpMultipleTransfer(String localFileName,
String localPath, String remoteRelativePath, String
remoteFileName, FtpPriority priority, Date
inQueueDateAndTime, long identifier){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier);
  }
```

```
  public LogicTuple getLogicTuple(){
    TupleArgument[] listaArgomenti=new TupleArgument[6];
    listaArgomenti[0]=new Value("S"+this.localFileName);
    listaArgomenti[1]=new Value("S"+this.localPath);
    listaArgomenti[2]=new Value("S"+this.remoteFileName);
    listaArgomenti[3]=new Value("S"+this.remoteRelativePath);
    listaArgomenti[4]=new
Value("S"+this.priority.priority());
    listaArgomenti[5]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.inQueueDateAndTi
me));
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }



  public LogicTuple getLogicTupleTemplate(){
    TupleArgument[] listaArgomenti=new TupleArgument[6];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Var();
    listaArgomenti[5]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }

  public LogicTuple getLogicTupleTemplate(FtpPriority
priority){
    TupleArgument[] listaArgomenti=new TupleArgument[6];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Value(priority.priority());
    listaArgomenti[5]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }

  public void parse(LogicTuple tuple) throws
InvalidTupleOperationException, ParseException{

    String temp="'";
    int start=1+1;

    temp=tuple.getArg(0).toString();
    temp=temp.substring(start,temp.length()-1);
    localFileName=temp;

    temp=tuple.getArg(1).toString();
    temp=temp.substring(start,temp.length()-1);
    localPath=temp;
```

```
    temp=tuple.getArg(2).toString();
    temp=temp.substring(start,temp.length()-1);
    remoteFileName=temp;

    temp=tuple.getArg(3).toString();
    temp=temp.substring(start,temp.length()-1);
    remoteRelativePath=temp;

    temp=tuple.getArg(4).toString();
    temp=temp.substring(start,temp.length()-1);
    priority=FtpPriority.parse(temp);

    temp=tuple.getArg(5).toString();
    temp=temp.substring(start,temp.length()-1);

inQueueDateAndTime=DateFormat.getDateTimeInstance(DateFormat.
LONG,DateFormat.LONG,Locale.getDefault()).parse((temp));

  }

}
                              ---
package tucsonftpmodel;

import java.text.*;
import java.util.*;
import ftpmodel.*;
import alice.logictuple.*;

public class TucsonFtpMultipleTransferDone extends
FtpMultipleTransferDone implements
ITucsonFtpMultipleTransferDone{

  private final String nomeTuplaLogica="ftpDone";

  public TucsonFtpMultipleTransferDone(){
    super();
  }

  public TucsonFtpMultipleTransferDone(String localFileName,
String localPath, String remoteRelativePath, String
remoteFileName, FtpPriority priority, Date
inQueueDateAndTime, long identifier, Date startDateAndTime,
Date stopDateAndTime){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier, startDateAndTime,
stopDateAndTime);
  }

  public LogicTuple getLogicTuple(){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Value("S"+this.localFileName);
```

```
    listaArgomenti[1]=new Value("S"+this.localPath);
    listaArgomenti[2]=new Value("S"+this.remoteFileName);
    listaArgomenti[3]=new Value("S"+this.remoteRelativePath);
    listaArgomenti[4]=new
Value("S"+this.priority.priority());
    listaArgomenti[5]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.inQueueDateAndTi
me));
    listaArgomenti[6]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.startDateAndTime
));
    listaArgomenti[7]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.stopDateAndTime)
);
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }


  public LogicTuple getLogicTupleTemplate(){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Var();
    listaArgomenti[5]=new Var();
    listaArgomenti[6]=new Var();
    listaArgomenti[7]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }

  public LogicTuple getLogicTupleTemplate(FtpPriority
priority){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Value(priority.priority());
    listaArgomenti[5]=new Var();
    listaArgomenti[6]=new Var();
    listaArgomenti[7]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }

   public void parse(LogicTuple tuple) throws
InvalidTupleOperationException, ParseException{

     String temp="'";
     int start=1+1;
```

```
     temp=tuple.getArg(0).toString();
     temp=temp.substring(start,temp.length()-1);
     localFileName=temp;

     temp=tuple.getArg(1).toString();
     temp=temp.substring(start,temp.length()-1);
     localPath=temp;

     temp=tuple.getArg(2).toString();
     temp=temp.substring(start,temp.length()-1);
     remoteFileName=temp;

     temp=tuple.getArg(3).toString();
     temp=temp.substring(start,temp.length()-1);
     remoteRelativePath=temp;

     temp=tuple.getArg(4).toString();
     temp=temp.substring(start,temp.length()-1);
     priority=FtpPriority.parse(temp);

     temp=tuple.getArg(5).toString();
     temp=temp.substring(start,temp.length()-1);

inQueueDateAndTime=DateFormat.getDateTimeInstance(DateFormat.
LONG,DateFormat.LONG,Locale.getDefault()).parse((temp));

     temp=tuple.getArg(6).toString();
     temp=temp.substring(start,temp.length()-1);

startDateAndTime=DateFormat.getDateTimeInstance(DateFormat.LO
NG,DateFormat.LONG,Locale.getDefault()).parse((temp));

     temp=tuple.getArg(7).toString();
     temp=temp.substring(start,temp.length()-1);

stopDateAndTime=DateFormat.getDateTimeInstance(DateFormat.LON
G,DateFormat.LONG,Locale.getDefault()).parse((temp));

   }

}
                              ---
package tucsonftpmodel;

import java.text.*;
import java.util.*;
import ftpmodel.*;
import alice.logictuple.*;

public class TucsonFtpMultipleTransferError extends
FtpMultipleTransferError implements
ITucsonFtpMultipleTransferError{

  private final String nomeTuplaLogica="ftpError";
```

```
  public TucsonFtpMultipleTransferError(){
    super();
  }

  public TucsonFtpMultipleTransferError(String localFileName,
String localPath, String remoteRelativePath, String
remoteFileName, FtpPriority priority, Date
inQueueDateAndTime, long identifier, Date startDateAndTime,
Date errorDateAndTime){

super(localFileName,localPath,remoteRelativePath,remoteFileNa
me,priority,inQueueDateAndTime,identifier, startDateAndTime,
errorDateAndTime);
  }

  public LogicTuple getLogicTuple(){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Value("S"+this.localFileName);
    listaArgomenti[1]=new Value("S"+this.localPath);
    listaArgomenti[2]=new Value("S"+this.remoteFileName);
    listaArgomenti[3]=new Value("S"+this.remoteRelativePath);
    listaArgomenti[4]=new
Value("S"+this.priority.priority());
    listaArgomenti[5]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.inQueueDateAndTi
me));
    listaArgomenti[6]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.startDateAndTime
));
    listaArgomenti[7]=new
Value("S"+DateFormat.getDateTimeInstance(DateFormat.LONG,Date
Format.LONG,Locale.getDefault()).format(this.errorDateAndTime
));
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }


  public LogicTuple getLogicTupleTemplate(){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Var();
    listaArgomenti[5]=new Var();
    listaArgomenti[6]=new Var();
    listaArgomenti[7]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }
```

```
  public LogicTuple getLogicTupleTemplate(FtpPriority
priority){
    TupleArgument[] listaArgomenti=new TupleArgument[8];
    listaArgomenti[0]=new Var();
    listaArgomenti[1]=new Var();
    listaArgomenti[2]=new Var();
    listaArgomenti[3]=new Var();
    listaArgomenti[4]=new Value(priority.priority());
    listaArgomenti[5]=new Var();
    listaArgomenti[6]=new Var();
    listaArgomenti[7]=new Var();
    return new LogicTuple(nomeTuplaLogica,listaArgomenti);
  }


  public void parse(LogicTuple tuple) throws
InvalidTupleOperationException, ParseException{

    String temp="'";
    int start=1+1;

    temp=tuple.getArg(0).toString();
    temp=temp.substring(start,temp.length()-1);
    localFileName=temp;

    temp=tuple.getArg(1).toString();
    temp=temp.substring(start,temp.length()-1);
    localPath=temp;

    temp=tuple.getArg(2).toString();
    temp=temp.substring(start,temp.length()-1);
    remoteFileName=temp;

    temp=tuple.getArg(3).toString();
    temp=temp.substring(start,temp.length()-1);
    remoteRelativePath=temp;

    temp=tuple.getArg(4).toString();
    temp=temp.substring(start,temp.length()-1);
    priority=FtpPriority.parse(temp);

    temp=tuple.getArg(5).toString();
    temp=temp.substring(start,temp.length()-1);

inQueueDateAndTime=DateFormat.getDateTimeInstance(DateFormat.
LONG,DateFormat.LONG,Locale.getDefault()).parse((temp));

    temp=tuple.getArg(6).toString();
    temp=temp.substring(start,temp.length()-1);

startDateAndTime=DateFormat.getDateTimeInstance(DateFormat.LO
NG,DateFormat.LONG,Locale.getDefault()).parse((temp));

    temp=tuple.getArg(7).toString();
```

```
    temp=temp.substring(start,temp.length()-1);

errorDateAndTime=DateFormat.getDateTimeInstance(DateFormat.LO
NG,DateFormat.LONG,Locale.getDefault()).parse((temp));

  }

}
```

Package tucsonmodel

```
package tucsonmodel;

import java.text.*;
import ftpmodel.*;
import alice.logictuple.*;

public interface ITucsonConverter{

  public LogicTuple getLogicTuple();
  public LogicTuple getLogicTupleTemplate();
  public LogicTuple getLogicTupleTemplate(FtpPriority
priority);
  public void parse(LogicTuple tuple) throws
InvalidTupleOperationException, ParseException;

}
```