Indice

1 Introduzione

La necessità di reperire e scambiare informazioni in modo sicuro ed efficiente è sempre stato il motore che ha alimentato lo sviluppo delle reti. Numerosi, infatti, sono stati i protocolli di comunicazione di rete sviluppati per il trasferimento di file.

Considereremo, in particolare, HTTP (Hypertext Transfer Protocol) che, a partire dal 1990 è diventato il protocollo di trasferimento del web, ed FTP (File Transfer Protocol) che ha un origine più remota – il primo documento ufficiale è del 1973 – e che si è evoluto nel corso degli anni come protocollo affidabile ed efficiente.

Entrambe i protocolli adottano per la comunicazione il modello client/server: lo scopo di questa tesi è di estendere l'architettura del sistema in modo da supportare servizi avanzati per lo scaricamento di file, permettendo così anche di fornire all'utente una visione integrata ed uniforme del sistema, nonostante l'impiego di protocolli differenti.

A questo fine ci si propone di introdurre nella classica architettura client/server un livello intermedio, costituito dall'infrastruttura di coordinazione TuCSoN, un'infrastruttura per la coordinazione di agenti su Internet basata su un'astrazione di comunicazione programmabile.

Questo canale programmabile, detto centro di tuple, è uno spazio di interazione in cui sono contenute tuple logiche che racchiudono informazione. La possibilità di programmare, in modo trasparente agli agenti, il comportamento di un centro di tuple in risposta ad un evento di comunicazione ha un'importanza fondamentale. Tale programmazione infatti, che viene specificata sempre nella forma di tuple logiche tramite il linguaggio ReSpecT, consente, unitamente all'impiego di agenti, di

realizzare le politiche di coordinazione più opportune per fornire la fruizione integrata dei servizi.

A questo fine ci si propone di sviluppare due componenti software, uno per il trasferimento di file tramite HTTP, l'altro tramite FTP, che dovranno soddisfare i seguenti requisiti:

- Assicurare compatibilità con qualsiasi server standard HTTP o FTP, da cui effettuare le operazioni di download.
- Basarsi sull'infrastruttura di coordinazione TuCSoN, realizzando opportuni centri di tuple in cui depositare le informazioni relative alle operazioni richieste o effettuate.
- Impiegare agenti TuCSoN che fungano da entità intermediarie per lo scambio di informazioni fra l'infrastruttura di coordinazione da un lato e il server o l'utente dall'altro (figura 1.1).

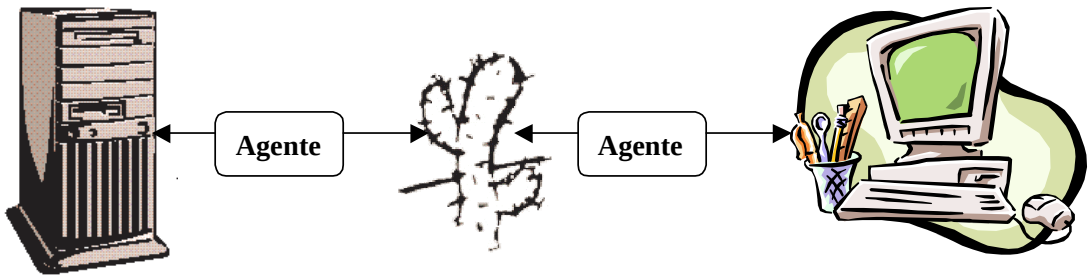


Figura 1.1 Modello di interazione nel sistema

Si noti come la struttura considerata sia relativa ad un'applicazione che sfrutta indifferentemente uno dei due protocolli, e preveda quindi la possibilità di una facile integrazione dei due servizi.

2 Il protocollo HTTP

L'Hypertext Transfer Protocol (HTTP) è un protocollo di livello applicativo per i sistemi distribuiti. HTTP è stato adottato dalla "World Wide Web Global Information Iniziative" a partire dal 1990. La prima versione HTTP/0.9 era un semplice protocollo per un trasferimento grezzo di dati in Internet. La versione successiva HTTP/1.0 ha migliorato il protocollo realizzando, per i messaggi, un formato simile a quello usato per la posta elettronica come definito dal Multipurpose Internet Mail Extensions (MIME). I messaggi contengono perciò meta-informazioni sui dati trasferiti.

Infine la versione HTTP/1.1 ha sviluppato problematiche come la realizzazione di supporti per connessioni persistenti e per il caching.

HTTP si basa sul modello di comunicazione Request/Response, ovvero un client invia una richiesta al server in attesa di una sua risposta. Il protocollo fornisce un insieme aperto di metodi al fine di indicare lo scopo di una richiesta e definisce inoltre la struttura dei riferimenti attraverso l'Uniform Resource Identifier (URI) come locazione (URL) o nome (URN) in modo da indicare la risorsa a cui il metodo deve essere applicato.

HTTP è anche usato come protocollo generico per la comunicazione tra agenti dell'utente e proxy o gateway verso altri sistemi Internet, inclusi quelli supportati da altri protocolli come SMTP, NNTP, FTP, Gopher. In questo modo HTTP consente l'accesso a risorse disponibili su diverse applicazioni.

Per dare una definizione formale del protocollo HTTP utilizzeremo la sintassi della forma estesa di Backus Naur.

2.1 Uniform Resource Identifier

Un URI è conosciuto con molti nomi: indirizzo WWW, Universal Document Identifier, Uniform Resource Locator (URL) o Uniform Resource Name (URN). Per quanto concerne il protocollo HTTP, un URI è una semplice stringa con un particolare formato che identifica, attraverso la locazione, il nome o qualsiasi altra caratteristica, una risorsa.
Lo schema di un URL HTTP secondo la notazione BNF è il seguente:

HTTP_URL *= "http:" "//" host [":" port] [abs_path ["?" query]]*

La risorsa risiede sul server in ascolto sulla porta dell'host indicato, mentre *abs_path* è un URI che identifica la risorsa stessa.
Se la porta non è specificata, si assume la porta 80. Inoltre se *abs_path* non è presente nell'URL, deve essere sostituito con "/" quando è usato come URI di una *request*.

2.2 Il messaggio HTTP

Essendo HTTP un protocollo di tipo Request/Response, sia la richiesta che la risposta HTTP hanno due sezioni principali: un message-header, che contiene tutte le informazioni relative alla richiesta (o risposta) e un message-body che contiene eventuali entità trasportate dal pacchetto, nonché tutte le informazioni ad esso relative (ad esempio mime-type, dimensioni, ecc.).

Le informazioni contenute all'interno di un messaggio HTTP sono separate mediante la sequenza di caratteri "CR/LF" dove CR è il carattere "Carriage Return" (ASCII 13) e LF è il carattere "Line Feed" (ASCII 10).
Secondo la sintassi BNF, il messaggio HTTP è così definito:

HTTP –Message *= Request | Response*

Request *= generic-message*
Response *= generic-message*

generic-message *= start-line*
**(message-header CRLF)*
CRLF
[message-body]

start-line *= Request-line | Response-line*

2.2.1 Request

Una richiesta HTTP da un client ad un server è definita dalla regola BNF:

Request *= Request-line*
**((general header | request-header | entity-header) CRLF)*
CRLF
[message-body]

La *request-line* contiene il metodo che deve essere applicato alla risorsa, l'identificatore della risorsa, e la versione del protocollo in uso.

Request-Line *= Method SP Request-URI SP HTTP-Version CRLF*
dove:

SP *= <US-ASCII Space (32)>*

Method *= "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT"*
| "DELETE" | "TRACE" | "CONNECT" | extension-method

extension-method *= token*

Request-URI *= "*" | absoluteURI | abs-path | authority*

HTTP-version *= "HTTP" "|" 1*DIGIT "." 1*DIGIT*

DIGIT *= <any US ASCII digit "0"..."9">*

I metodi saranno trattati in dettaglio in seguito. Per quanto riguarda *Request-Uri*, la scelta tra le quattro opzioni specificate dipende dalla natura della richiesta. In particolare l'asterisco "*" indica che la richiesta non è applicata ad una particolare risorsa ma al server stesso.
La forma con *absoluteURI* è necessaria quando la richiesta viene fatta ad un proxy, che può inoltrarla ad un altro proxy o direttamente al server, o servirla esso stesso.
La forma più comune è quella in cui il *Request-URI* contiene il percorso assoluto e la locazione dell'URI nella rete viene trasmessa in un campo header relativo all'host.
Il campo *General-header* ha valore generico per entrambi i messaggi di richiesta o risposta e non contiene dati applicabili alla entità trasportata con il messaggio:

General-header *= Cache-Control | Connection | Date | Pragma | Trailer*
| Transfer-Encoding | Upgrade | Via | Warning

Senza scendere nei dettagli di ogni singolo campo, si inferisce come il General-header trasporti le informazioni necessarie alla negoziazione tra le applicazioni. Ad esempio, il campo *Connection* può essere utilizzato da un

server proxy per determinare lo stato della connessione tra client e server e decidere su eventuali misure da applicare.
I campi di *request-header* consentono al client di inviare informazioni relative alla richiesta in corso e al client stesso.

Request-header = *Accept* | *Accept-Charset* | *Accept-Encoding*
| *Accept-Language* | *Authorization* | *Expect* | *From*
| *Host* | *If-Match* | *If-Modified-Since* | *If-None-Match*
| *If-Range* | *If-Unmodified-Since* | *Max-Forwards*
| *Proxy-Authorization* | *Range* | *Referer* | *TE*
| *User-Agent*

2.2.2 Response

Dopo aver ricevuto un messaggio di richiesta, il server risponde con un messaggio di risposta definito dalla regola BNF:

Response = *Status-line*
**((general header | response-header | entity-header) CRLF)*
CRLF
[message-body]

La *Status-line* è la prima linea di un messaggio di risposta HTTP e consiste in tre token separati da spazio contenenti rispettivamente informazioni sulla versione del protocollo, un codice di stato (Status-Code) che indica un errore o l'eventuale buon fine della richiesta, una breve descrizione del codice di stato (Reason-Phrase).

Status-line *= HTTP-Version SP Status-Code SP Reason-Phrase CRLF*

SP *= <Space>*

Status-Code ***=*** *"100" : Continue*

| "101": Switching Protocols

| "200": OK

| extension-code

...etc...

extension-code *= 3DIGIT*

Reason-Phrase *= *<TEXT, excluding CR, LF>*

La prima cifra di un codice di stato indica il tipo di codice:

1xx : Informativo – Richiesta ricevuta, continua l'analisi dei dati.

2xx : Successo – L'azione è stata ricevuta con successo, compresa ed accettata.

3xx : Ridirezione – Devono essere compiute ulteriori azioni al fine di completare la richiesta.

4xx : Client-Error – La richiesta è sintatticamente errata e non può essere soddisfatta.

5xx : Server-Error – Il server non ha soddisfatto la richiesta, pur sintatticamente valida.

Come per il *Request-Header*, il *Response-Header* contiene campi utili alla trasmissione di informazioni aggiuntive relative alla risposta che non possono essere contenute nella *Status-Line*. Le informazioni sono relative al server e ad eventuali ulteriori accessi alla risorsa:

response-header = Accept-Ranges | Age | Etag | Location
| Proxy-Authenticate | Retry-After | Server | Vary
| www-Authenticate

2.2.3 Entity

Un'entità è costituita da un entity-header, ossia il campo descrittore della risorsa trasportata, e da un entity-body anche se, in alcuni casi, il messaggio di risposta può contenere solamente il campo entity-header.
L'entity-header trasporta meta-informazioni relative alla risorsa identificata dalla richiesta.

entity-header = *Allow* | *Content-Encoding* | *Content-Language*
| *Content-Lenght* | *Content-Location* | *Content-MD5*
| *Content-Range* | *Content-Type* | *Expires*
| *Last- Modified* | *extension-header*
extension-header = *message-header*

Queste informazioni, a seconda dei casi, possono essere necessarie ed opzionali. Il campo extension-header consente l'aggiunta di nuove informazioni non previste, ma necessarie, senza apportare modifiche al protocollo.
L'entity-body è presente solo se è presente il message-body ed è da questo ottenuto decodificando ogni codifica di trasferimento (Transfer-Encoding) che è stata applicata al messaggio. Il tipo di dati trasferito è determinato dai campi header Content-Encoding, che specifica l'eventuale codifica dei dati, e Content-Type, che indica il tipo di dati. L'entity-body in sè risulterà quindi definito come:

entity-body *= *OCTET*

OCTET *= <any 8-bit sequence of data>*

2.3 I metodi HTTP

Il protocollo fornisce un insieme di metodi al fine di indicare lo scopo di una richiesta. I metodi principali sono:

- **OPTIONS**: rappresenta una richiesta di informazioni sulle opzioni di comunicazione e consente al client di determinare le opzioni e/o i requisiti associati ad una risorsa senza implicare un'azione sulla risorsa stessa.
- **GET**: recupera una qualsiasi informazione, nella forma di entità, identificata da un Request-URI.
- **HEAD**: è identico al metodo GET, ma il messaggio di response non conterrà il message-body. E' quindi usato per ottenere meta-informazioni sull'entità senza trasferire l'entità stessa.
- **POST**: è usato per richiedere che il server accetti l'entità racchiusa nella request come subordinata alla risorsa identificata dal Request-URI. Svolge quindi funzioni quali l'annotazione di risorse esistenti, inserire un messaggio in una mailing list o in un newsgroup, fornire ad un processo i dati contenuti in una form, estendere un database attraverso operazioni di append.
- **PUT**: richiede che l'entità racchiusa sia allocata al Request-URI indicato. Nel caso il Request-URI non riferisca una risorsa già esistente, il server può crearla con l'URI indicato, altrimenti, l'entità

racchiusa nella request dovrebbe essere considerata come una versione più aggiornata di quella già esistente.

- **DELETE**: richiede che il server elimini la risorsa identificata dal Request-URI. Tale metodo potrebbe però essere stato ridefinito sul server, in modo che il client non è garantito che la risorsa sia effettivamente eliminata anche se, in caso di status-code indicante successo, essa dovrebbe essere resa inaccessibile.
- **TRACE**: è usato per invocare un loop-back remoto di un messaggio request. Consente quindi al client di testare che cosa viene effettivamente ricevuto dal server o da un proxy intermedio.
- **CONNECT**: tale metodo è riservato per essere usato con un proxy che può dinamicamente essere trasformato in un tunnel.

3 Il protocollo FTP

Il File Transfer Protocol (FTP) ha avuto una lunga evoluzione nel corso degli anni. Il primo meccanismo per il trasferimento di file fu sviluppato nel 1971 dal M.I.T. (Massachussets Institute of Technology). Successivamente FTP fu definito come protocollo per il trasferimento di file tra gli hosts di ARPANET in modo sicuro ed efficiente. Nel 1973 l'RFC 454 costituì il primo documento ufficiale sulle specifiche di questo protocollo. Da allora sono state apportate considerevoli modifiche al protocollo e sono state aggiunte numerose funzionalità anche se la struttura generale è rimasta la stessa.

Il fine di FTP è quello di agevolare la condivisione di file attraverso l'accesso anche implicito, cioè attraverso l'uso di programmi appositi, a macchine remote, garantendo un trasferimento di dati affidabile ed efficiente e proteggendo l'utente da variazioni nei sistemi di memorizzazione dei file tra gli host.

3.1 Il modello FTP

Come si vede dalla figura 3.1, il modello FTP prevede uno o più processi, sia lato server che lato utente, che operano su due distinte connessioni: una di controllo e una per il trasferimento dati.

La connessione di controllo, che segue il protocollo Telnet, è il canale attraverso cui l'interprete del protocollo lato utente invia comandi al corrispondente processo lato server e riceve da questo risposte positive o negative a seconda del risultato dell'operazione. In generale è l'user PI che svolge il ruolo attivo iniziando la connessione di controllo, da una sua porta verso il server PI, che era in ascolto su una data porta.

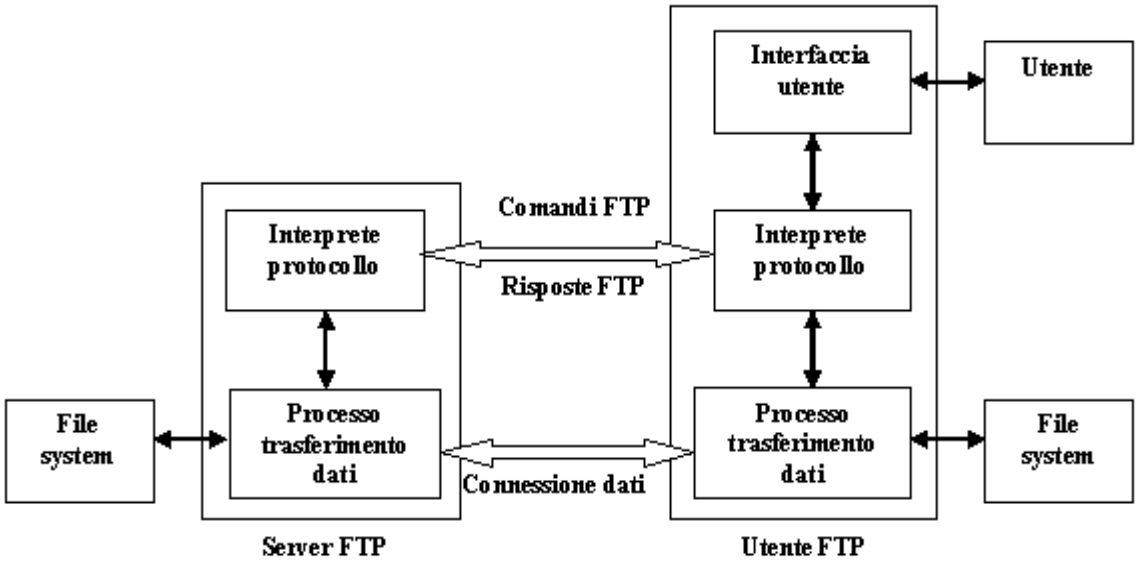


Figura 3.1 Il modello FTP

Una volta stabilita la connessione di controllo, i comandi FTP inviati dall'utente specificano i parametri per la connessione dati, come ad esempio la porta per la connessione dati, il modo di trasferimento o la rappresentazione dei dati trasmessi, e la natura dell'operazione di trasferimento, come recupero di un file, suo salvataggio o sua cancellazione.

Il processo per il trasferimento dati lato utente è in ascolto su una porta ed è il corrispondente processo lato server che, in accordo con i parametri suddetti passati dal proprio interprete di protocollo, inizia la connessione dati.

Il protocollo richiede che la connessione di controllo rimanga aperta durante il trasferimento dei dati: è responsabilità dell'utente richiederne la chiusura quando termina l'impiego di servizi FTP. Il server può interrompere il trasferimento dei dati se la connessione di controllo viene chiusa senza un comando.

La connessione dati può essere usata in entrambe le direzioni anche per un invio e una ricezione simultanei. Inoltre non è detto che essa esista sempre, poichè alcune operazioni FTP non la richiedono. In generale è responsabilità del server mantenere la connessione dati e chiuderla, ad eccezione dei casi in cui l'user DTP sta inviando dati in una modalità di trasferimento che richiede che la connessione sia chiusa indicando End-Of-File (EOF).
Si ricorda infine che il protocollo FTP non richiede che la porta in ascolto per la connessione dati sia sullo stesso host che ha iniziato la connessione di controllo, anche se deve essere garantito che su quella porta vi sia un processo in ascolto.

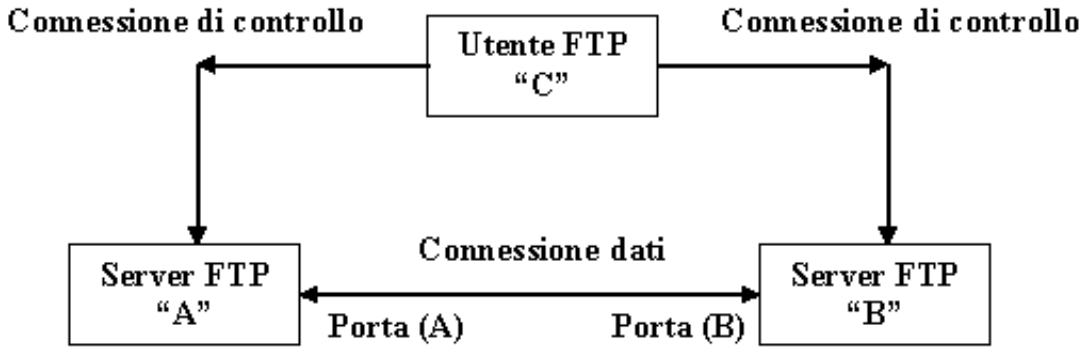


Figura 3.2 Trasferimento file tra host non locali

Si potrebbe, ad esempio, voler trasferire un file tra due hosts entrambe non locali. L'utente allora stabilisce una connessione di controllo con i due server e dispone per una connessione dati tra i due, come mostrato in fig. 3.2.

3.2 Trattamento dei dati

Il protocollo FTP prevede la trasmissione di dati tra sistemi eterogenei che hanno rappresentazioni dei dati differenti. E' dunque necessario effettuare trasformazioni dei dati.

FTP adotta lo standard Network Virtual Terminal (NVT), come definito nel protocollo Telnet. Sulla rete si definisce un unico terminale virtuale e in corrispondenza di ogni host vengono effettuate le opportune conversioni da terminale virtuale a terminale locale e viceversa.

L'utente gestisce la rappresentazione dei dati definendo un tipo di rappresentazione che determina una dimensione logica dei byte che va comunque distinta, come detto, dalla dimensione di transferimento dei byte (che è sempre di 8 bits).

I tipi di dato per FTP sono:

- **ASCII**: è il tipo di default e deve essere accettato da tutte le implementazioni di FTP. E' usato per il trasferimento di file testuali. Il mittente converte quindi i dati da una rappresentazione interna alla rappresentazione standard 8-bit NVT-ASCII; il ricevente convertirà i dati dallo standard alla sua rappresentazione interna.
- **EBCDIC**: è usato per trasferimenti tra host che usano questo EBCDIC per la loro rappresentazione interna di caratteri. Nella trasmissione i dati sono rappresentati come caratteri EBCDIC a 8 bit. La codifica dei caratteri è l'unica cosa che distingue i tipi EBCDIC e ASCII.
- **IMAGE**: i dati sono inviati come bit contigui che, per il trasferimento, sono impacchettati nei byte di trasferimento.
- **LOCAL**: i dati sono trasferiti in byte logici di dimensioni che devono essere obbligatoriamente specificate. Se la dimensione non corrisponde a quella di trasferimento i byte logici devono essere

impacchettati in modo contiguo senza tenere conto delle dimensioni del byte di trasferimento.

Oltre ai differenti tipi di rappresentazione, FTP consente di specificare la struttura di un file. Sono definiti tre tipi di struttura:

1. **File-structure**: non vi è struttura interna e il file è considerato come una sequenza continua di byte.
2. **Record-structure**: il file è costituito da record sequenziali.
3. **Page-structure:** il file è costituito da pagine indipendenti indicizzate.

Inoltre è possibile scegliere il modo di trasmissione dei dati. Sono definiti tre tipi di trasmissione:

1. **Stream mode**: i dati sono trasmessi come stream di byte.
2. **Block mode**: il file è trasmesso come una serie di blocchi di dati preceduti da uno o più byte di intestazione che contengono un campo contatore, che indica la lunghezza del blocco, e un codice descrittore che contiene informazioni come l'ultimo blocco del file o dati in sospetto di errore.
3. **Compressed mode**: è previsto l'invio di tre generi di informazioni. Dati regolari, inviati in una stringa di byte, dati compressi, quando si hanno replicazioni dello stesso byte o riempimenti, e informazioni di controllo inviati in una sequenza di due byte di cui il primo ha tutti i bit a 0 e il secondo contiene un codice descrittore come quello descritto per il block mode.

3.3 Funzioni per il trasferimento dei file

L'interprete di protocollo lato utente è responsabile dell'invio di comandi FTP al corrispondente processo lato server che li interpreta, invia risposte e all'occorrenza comanda al proprio DTP di stabilire la connessione e trasferire i dati.
Vengono qui di seguito analizzati i principali comandi FTP.

- **USER**: ha come argomento una stringa Telnet che identifica l'utente ed è normalmente il primo comando inviato una volta stabilita la connessione di controllo, poiché l'identificazione dell'utente è richiesta per accedere al file system del server.
- **PASS**: ha come argomento una stringa Telnet che specifica la password dell'utente. Ovviamente questo comando seguirà immediatamente il precedente per completare l'identificazione.
- **ACCT**: ha come argomento una stringa Telnet che identifica l'account dell'utente. Non è necessariamente in relazione con il comando USER; in alcuni casi è richiesto un account per il login o per specifici tipi di operazioni.
- **REIN**: il comando cancella tutte le informazioni di account di un utente e interrompe il flusso di I/O a meno che un trasferimento di file non sia in atto. La connessione di controllo viene lasciata aperta.
- **QUIT**: il comando termina la sessione di un utente chiudendo anche la connessione di controllo, a meno che un trasferimento di file non sia in atto. In tal caso la connessione di controllo viene chiusa al termine del trasferimento dopo aver inviato una risposta all'utente.

- **PORT**: ha come argomento una specifica di porta di un host che indica la porta da utilizzare nella connessione dati. Quando tale comando non viene utilizzato, viene impiegata la porta di default.
- **PASV**: chiede al processo di trasferimento dati lato server di svolgere un ruolo passivo e di mettersi in ascolto su una porta dati (non è quella di default). La risposta a tale comando contiene gli indirizzi dell'host e della porta su cui il server sta ascoltando.
- **TYPE**: specifica il tipo di rappresentazione dei dati.
- **STRU**: ha come argomento un singolo carattere che specifica la struttura del file.
- **MODE**: ha come argomento un singolo carattere che specifica il modo di trasferimento dei dati del file.
- **ABOR**: il server deve interrompere l'esecuzione del servizio del precedente comando FTP. Se l'esecuzione è già terminata non viene effettuata alcuna azione. La connessione di controllo rimane aperta, la connessione dati viene chiusa.
- **RETR**: viene trasferita una copia del file, specificato nel percorso indicato, dal server all'altro capo della connessione.
- **STOR**: trasferisce un file sul server. Se il file è già esistente sul server esso viene sostituito dai dati trasferiti, altrimenti il file viene creato.
- **APPE**: trasferisce un file sul server. Se il file è già esistente sul server i dati trasferiti vengono aggiunti al file, altrimenti il file viene creato.
- **DELE**: il file specificato viene cancellato sul server.
- **MKD**: viene creata la directory specificata nel percorso.
- **RMD**: viene rimossa la directory specificata nel percorso.

- **LIST**: viene inviata una lista all'utente che contiene la lista dei file di una directory se nel percorso è specificata una directory, informazioni sul file se nel percorso è indicato un file. Se il comando viene passato senza argomenti si fa riferimento alla directory di default.
- **HELP**: il server invia informazioni di aiuto per l'utente.
- **NOOP**: specifica una non-azione e attende una risposta di OK dal server.

Ogni comando FTP genera una o più risposte. Una risposta FTP è costituita da tre cifre (xyz) seguite da un testo esplicativo. Le tre cifre hanno ognuna un significato speciale. La prima cifra indica la bontà dell'operazione eseguita:

1yz : Risposta positiva preliminare: l'azione richiesta è iniziata. L'utente deve attendere un'ulteriore risposta prima di immettere un nuovo comando.

2yz : Risposta positiva di completamento: l'azione richiesta è stata completata con successo.

3yz : Risposta positiva intermedia: il comando è stato accettato ma l'azione richiesta necessita l'invio di ulteriori informazioni.

4yz : Risposta negativa transitoria di completamento: il comando non è stato accettato, ma la condizione di errore è temporanea è l'azione può essere richiesta nuovamente.

5yz : Risposta negativa permenente di completamento: il comando non è stato accettato e l'azione richiesta non è stata effettuata. L'utente è scoraggiato dal ripetere la richiesta.

La seconda cifra indica a che cosa si riferisce la risposta:

x0z : Sintassi
x1z : Informazioni
x2z : Connessioni
x3z : Autenticazione
x4z : Ancora non specificato
x5z : File System

Infine la terza cifra specifica meglio il tipo di risposta.

4 L’infrastruttura di coordinazione TuCSoN

La crescente necessità di accedere ed elaborare dinamicamente informazioni eterogenee distribuite in Internet ha richiesto l’elaborazione di nuovi modelli e paradigmi per lo sviluppo di applicazioni.
TuCSoN è un modello di coordinazione per applicazioni Internet basate su agenti mobili. Il modello è basato sulla nozione di centro di tuple, ossia uno spazio di interazione basato su tuple logiche. Una tupla logica è una struttura dati costituita da un nome e da una lista di argomenti.
Un centro di tuple è associato ad un nodo e può essere impiegato sia per la cooperazione tra agenti, sia per l’accesso a risorse locali.
In particolare, è possibile programmare, in modo trasparente agli agenti, il comportamento di un centro di tuple in risposta ad un evento di comunicazione, consentendo così di gestire in modo opportuno la dinamicità ed eterogeneità delle informazioni, assicurando integrità dei dati.
Una caratteristica fondamentale di TuCSoN è quella di fornire una duplice interpretazione dello spazio di coordinazione che può essere visto sia come risorsa locale sia come risorsa globale. Come risorsa globale, un centro di tuple è accessibile da ogni punto della rete, specificandone il nome e l'indirizzo; come risorsa locale, invece, un centro di tuple è identificato unicamente dal nome. Questa caratteristica si adatta ad entrambe i ruoli degli agenti Internet, ossia agenti locali o entità network-aware.

4.1 Interazione e accesso a TuCSoN

Dato un centro di tuple, è possibile interagire con esso, attraverso l’uso di primitive, inserendo o recuperando tuple o modificando specifiche di

comportamento a fronte di determinate azioni, tramite il linguaggio ReSpecT.
L'insieme delle primitive è costituito da:

out: inserisce la tupla specificata nel centro di tuple.

es: se si desidera inserire la tupla *p(1,hello)* l'istruzione sarà:

out*(p(1,hello))*

in: rimuove una tupla che soddisfa il template della tupla specificata.

es: se si desidera rimuovere una generica tupla che abbia come nome *p* e come secondo e ultimo argomento *hello* l'istruzione sarà:

in*(p(_,hello))*

la tupla dell'esempio precedente soddisferebbe questo template.

rd: legge una tupla che soddisfa il template della tupla specificata.

es: se si inserisce il comando

rd*(p(X,Y))*

viene letta una tupla di nome *p* con 2 argomenti i cui valori vengono immessi nelle variabili X e Y.

inp: rimuove, se presente, una tupla che soddisfa il template della tupla specificata (se la tupla non è presente l'operazione fallisce).

rdp: legge, se presente, una tupla che soddisfa il template della tupla specificata (se la tupla non è presente l'operazione fallisce).

set_spec: specifica il comportamento del il centro di tuple. La tupla deve essere un programma ReSpecT valido nella forma di stringa atomica o di lista di reazioni.

es: il comando

set_spec*(‘reaction(out(X),(out_r(backup(X))))’)*

programma il comportamento del centro di tuple in modo che sia immessa una tupla di backup per ogni nuova tupla inserita.

get_spec: ritorna il comportamento del centro di tuple.

I comandi, come indicati negli esempi, fanno riferimento al centro di tuple di default locale. Se le primitive devono essere applicate ad un altro centro di tuple è necessario specificarne il nome e l’indirizzo nella forma

TCName @ TCAddress ? op(Tuple)

L’interazione con l’infrastruttura può essere realizzata tramite l’interfaccia grafica CLIAgent (fig. 4.1), che consente di invocare direttamente le primitive su di un dato centro di tuple residente su un dato nodo, oppure attraverso applicazioni scritte in Java o in tuProlog (l’ambiente Prolog fornito dall’infrastruttura).

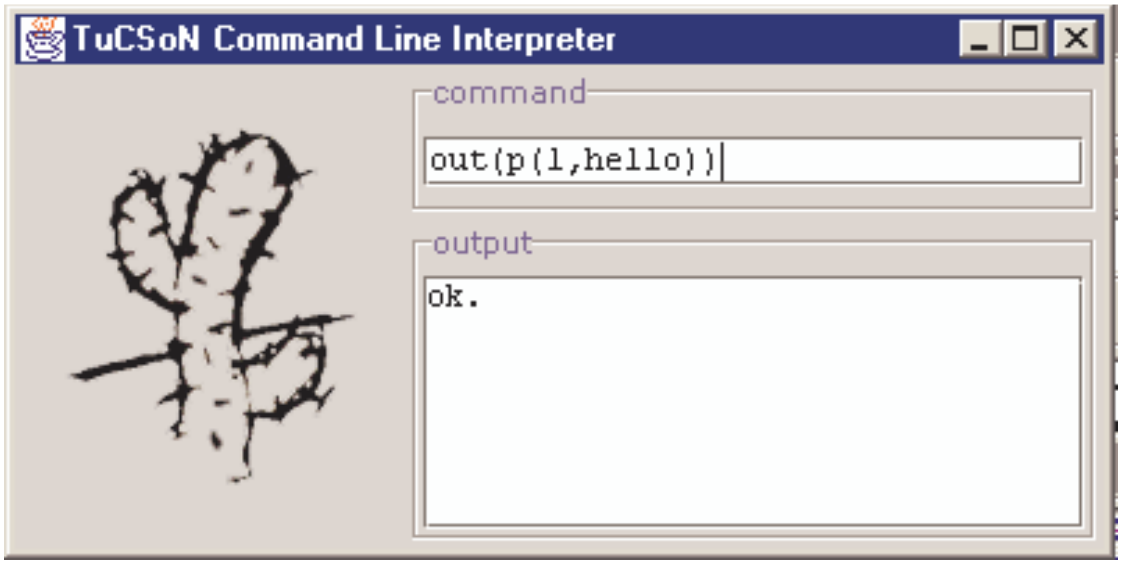


Figura 4.1 Interfaccia grafica CLIAgent

Per accedere all'infrastruttura TuCSoN deve essere acquisito e usato un contesto di coordinazione di agente (ACC).

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.

5 L’architettura del sistema

5.1 L’architettura classica e il nuovo approccio

L’architettura su cui si basano HTTP ed FTP è di tipo client/server. Tale modello di interazione prevede le seguenti fasi di comunicazione:

1. Il cliente inoltra una richiesta al server
2. Il server elabora la richiesta
3. Il server restituisce il risultato dell’elaborazione al client

La figura 5.1 illustra, in un’operazione di download di file, come entrambe i protocolli prevedano l’esecuzione delle fasi considerate.

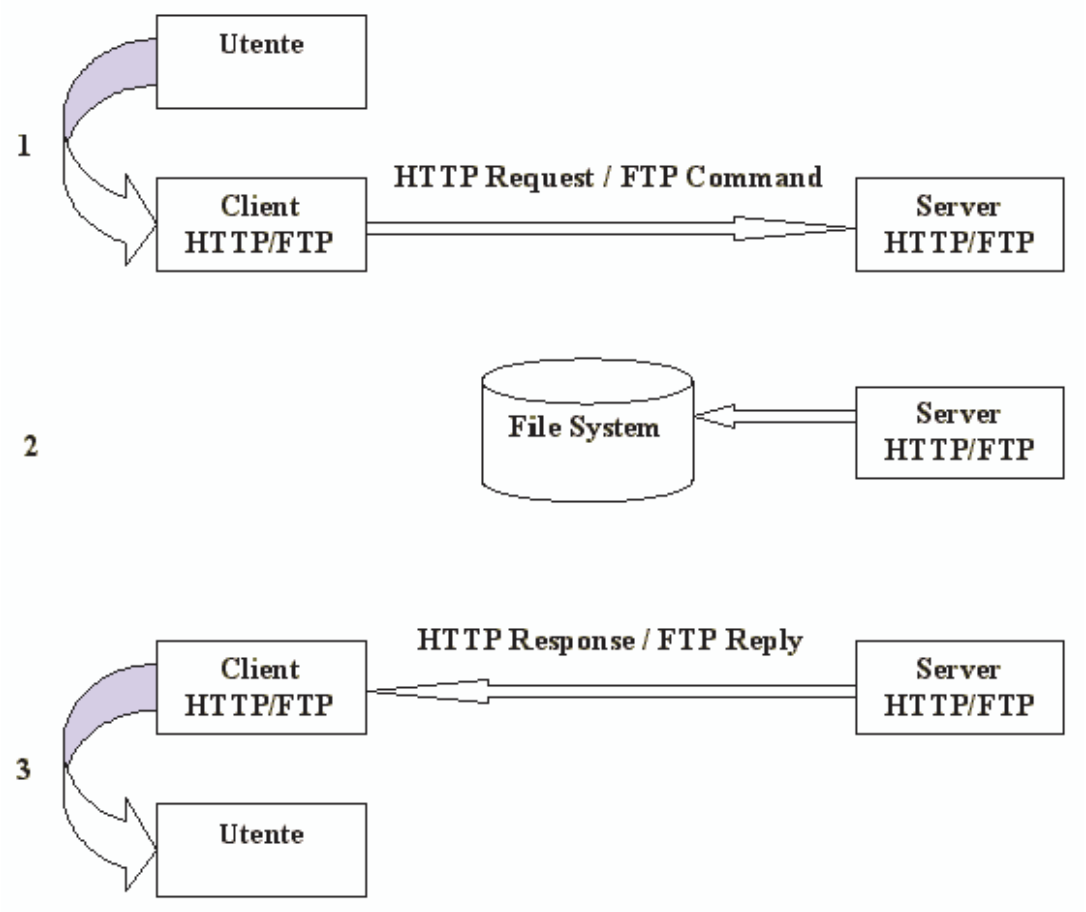
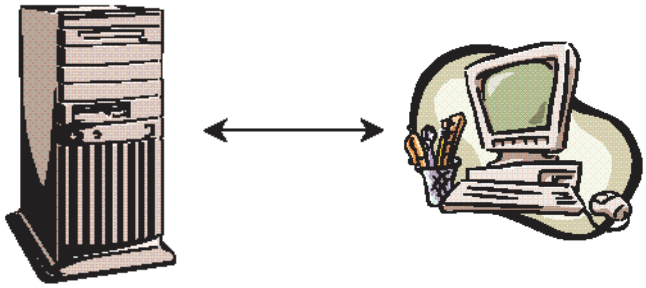
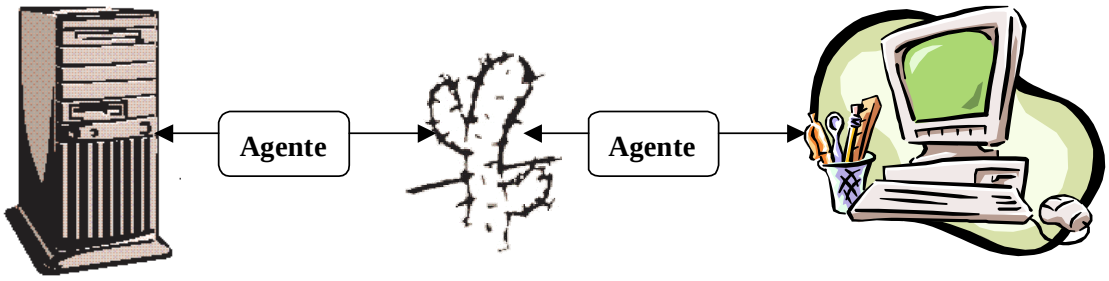


Figura 5.1 Download di file con HTTP/FTP

L’obiettivo è di estendere tale architettura in modo da realizzare un sistema intelligente per lo scaricamento di file che possa supportare servizi avanzati, e fornisca all’utente una visione integrata ed uniforme del sistema, nonostante l’impiego di protocolli differenti.
Ci si propone quindi di introdurre nella classica architettura client/server un livello intermedio, costituito dall’infrastruttura di coordinazione TuCSoN (fig. 5.2).



L’architettura classica



La nuova architettura

Figura 5.2 Il livello intermedio TuCSoN

Tale infrastruttura infatti fornisce, tramite i centri di tuple, un’astrazione di comunicazione programmabile. La possibilità di programmare, in modo trasparente agli agenti, il comportamento di un centro di tuple in risposta ad un evento di comunicazione ha un’importanza fondamentale. Tale programmazione infatti consente, unitamente all’impiego di agenti che

fungono da intermediari nella comunicazione all'interno del sistema come mostrato in figura 5.2, di realizzare le politiche di coordinazione più opportune per fornire la fruizione integrata dei servizi.

6 Progetto del sistema

6.1 Scelte progettuali

Il progetto prevede la realizzazione di due componenti software, uno per il trasferimento di file tramite HTTP, l'altro tramite FTP. Essi dovranno soddisfare i seguenti requisiti:

- Assicurare compatibilità con qualsiasi server standard HTTP o FTP, da cui effettuare le operazioni di download.
- Basarsi sull'infrastruttura di coordinazione TuCSoN, realizzando opportuni centri di tuple in cui depositare le informazioni relative alle operazioni richieste o effettuate.
- Impiegare agenti TuCSoN che fungano da mediatori fra l'infrastruttura di coordinazione da un lato e il server o l'utente dall'altro

Poiché TuCSoN consente l'uso di un numero potenzialmente illimitato di centri di tuple si è scelto di tenere separate le informazioni, utilizzando tre centri di tuple:

- Un centro di tuple contiene le richieste effettuate dall'utente.

  La tupla contiene un argomento che identifica la risorsa che si desidera scaricare (URL per HTTP, nome del file sul server per FTP) e un argomento che identifica il percorso locale e il nome del file con cui si desidera memorizzarla.

  Nella tupla sono inoltre contenute la priorità (alta, media o bassa) con cui deve essere eseguita la richiesta e l'ora e la data in cui la richiesta è stata effettuata.

- Un centro di tuple contiene le informazioni relative alle operazioni di download di file effettuate.

  La tupla contiene un argomento che identifica la risorsa scaricata (URL per HTTP, nome del file sul server per FTP) e un argomento che identifica il percorso locale e il nome del file con cui è stata memorizzata.

  Nella tupla sono inoltre contenute l'ora di inizio e di fine del trasferimento.
- Un centro di tuple contiene tuple relative a download che non sono stati portati a termine causa il verificarsi di errori sulla connessione o nel reperimento dei file.

  La tupla contiene un argomento che identifica la risorsa che si è tentato di scaricare (URL per HTTP, nome del file sul server per FTP) e un argomento che identifica il percorso locale e il nome del file con cui avrebbe dovuto essere memorizzata.

  Nella tupla sono anche contenute l'ora e la data in cui è iniziato il tentativo di trasferimento e il messaggio di errore.

La coordinazione dei tre centri di tuple è realizzata mediante l'impiego di agenti TuCSoN. In particolare si è scelto di utilizzare:

- Un agente TuCSoN per inserire le richieste dell'utente nell'opportuno centro di tuple.
- Un agente TuCSoN che legga le richieste, effettui l'operazione di download del file dal server e inserisca una tupla di successo o di fallimento nell'opportuno centro di tuple
- Un agente TuCSoN che visualizzi le richieste ancora da servire e le operazioni di scaricamento effettuate o fallite.

6.1.1 Accesso a TuCSoN tramite Java

La scelta di sviluppare il progetto tramite il linguaggio di programmazione Java, ha comportato la necessità di utilizzare strutture opportune che consentissero di accedere a TuCSoN e di implementare gli agenti necessari alla realizzazione del sistema.

Per accedere all'infrastruttura TuCSoN deve essere acquisito e usato un contesto di coordinazione di agente (ACC), che definisce l'insieme delle azioni permesse. Il package *alice.tucson.api* contiene le classi necessarie per accedere alla infrastruttura, ed in particolare un ACC è rappresentato dall'interfaccia *TucsonContext* che fornisce i metodi con cui realizzare le primitive di comunicazione.

Tuttavia un ACC è pensato per essere utilizzato da un solo agente, mentre ai fini del progetto è necessario realizzare più agenti TuCSoN indipendenti. Per fare ciò è necessario utilizzare la classe *Agent*, estendendola e ridefinendone il metodo *body()* come riportato nell'esempio seguente:

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

class MyAgent extends Agent {
  protected void body(){
    try {
      TupleCentreId tid = new TupleCentreId("test_tc");
      out(tid, new LogicTuple("p",new Value("hello world")));
      LogicTuple t=in(tid, new LogicTuple("p",new Var("X"));
      System.out.println(t);
    } catch (Exception ex){}
  }
}

public class Test {
  public static void main(String[] args) throws Exception {
    AgentId aid=new AgentId("alice");
    new MyAgent(aid).spawn();
  }
```

```
}
```

Inoltre è necessario utilizzare il package *alice.logictuple* che contiene classi che modellano il linguaggio di comunicazione di TuCSoN.

6.2 Sviluppo del componente HTTP

Il componente che consente lo scaricamento di file tramite il protocollo HTTP è costituito da due agenti:

- **HttpGui**: consente all'utente di effettuare le proprie richieste e di visualizzare sia le richieste non ancora servite, sia le operazioni effettuate.
- **HttpAgent**: legge le richieste di un utente ed effettua il download dei file ad esse relativi.

Gli agenti utilizzano per lo scambio di informazioni opportuni centri di tuple. Per ogni utente vengono creati tre centri di tuple: uno per le richieste, uno per le operazioni effettuate con successo e uno per gli errori. I nomi sono assegnati unendo il nome dell'utente e il tipo di centro di tuple. Si ha quindi rispettivamente: `username_requests`, `username_downloads`, `username_errors`.

6.2.1 L'agente HttpGui

L'agente HttpGui presenta un'interfaccia grafica (fig. 6.1) tramite la quale è possibile effettuare richieste di download di file, specificando il proprio nome di utente, l'URL della risorsa che si desidera scaricare e percorso e nome del file locale. E' inoltre possibile specificare la priorità con cui deve essere servita la richiesta.

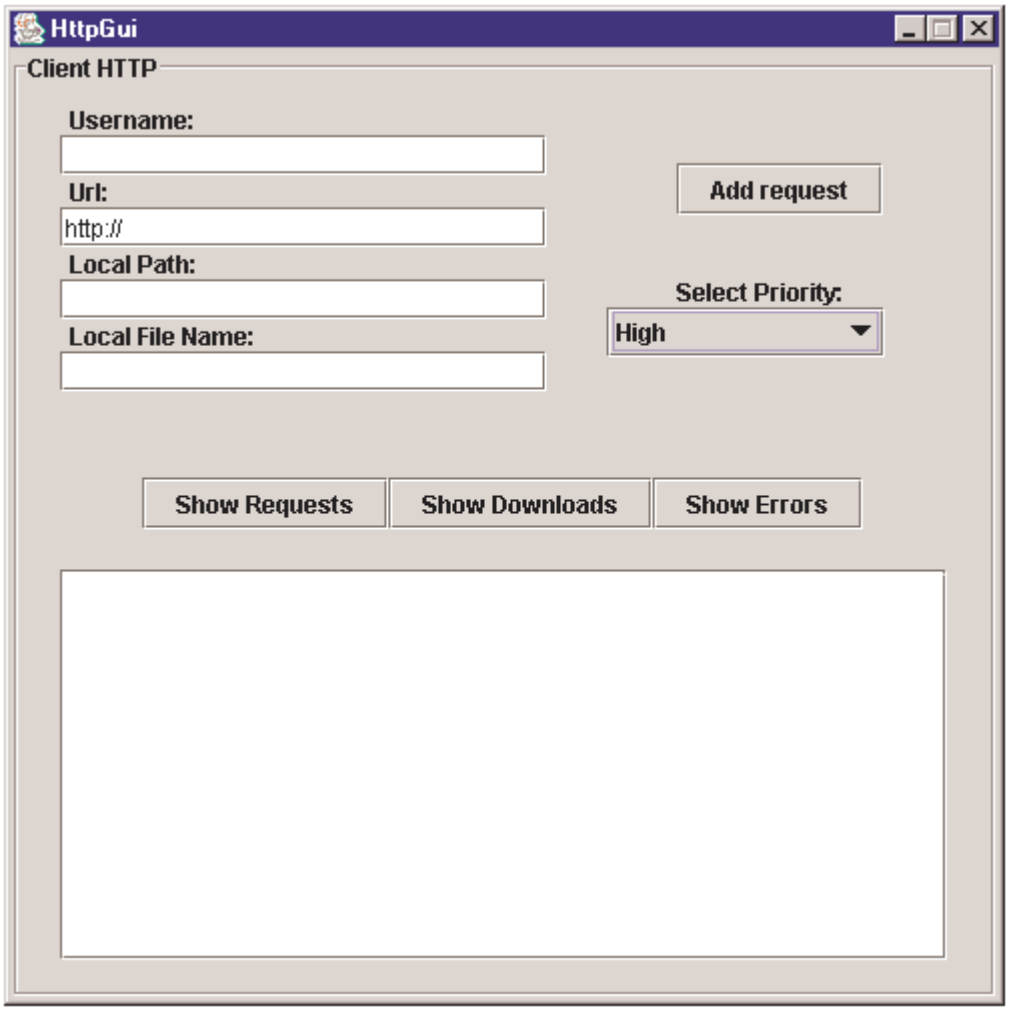


Figura 6.1 Interfaccia grafica dell'agente HttpGui

L'agente inserisce quindi nel centro di tuple `username_requests`, una tupla nella forma:

```
request(localPath,localName,url,priority,requestTime)
```

Quando si richiede di visualizzare l'insieme delle richieste o le operazioni effettuate o fallite, l'agente le legge dal corrispondente centro di tuple e le visualizza nell'area di testo.

6.2.2 L'agente HttpAgent

L’agente HttpAgent presenta una semplice interfaccia grafica (fig. 6.2) in cui l’utente, una volta inserito il proprio nome, può avviare le operazioni di scaricamento dei file.

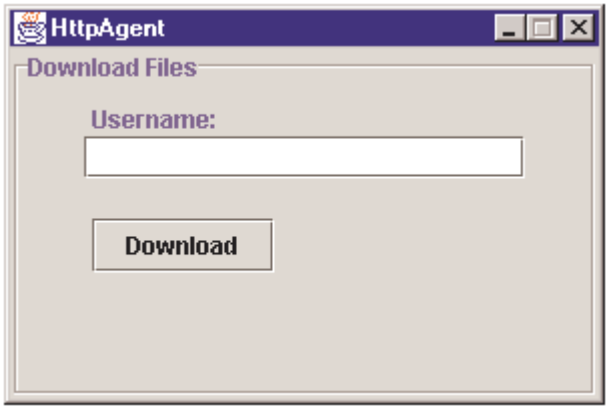


Figura 6.2 Interfaccia grafica dell'agente HttpAgent

L’agente legge una richiesta dal centro di tuple `username_requests`, scegliendola tra quelle con priorità più alta. A questo punto cerca di servire la richiesta e se l’operazione ha successo inserisce nel centro di tuple `username_downloads`, una tupla nella forma:

```
download(from,to,startTime,finishedTime)
```

Se invece l’operazione fallisce, l’agente inserisce nel centro di tuple identificato da `username_errors`, una tupla nella forma:

```
error(from,to,startTime,errorMsg)
```

Quando non sono presenti richieste da servire, l’agente rimane in attesa.

6.3 Sviluppo del componente FTP

Il componente che consente lo scaricamento di file tramite il protocollo FTP è costituito da due agenti:

- **FtpGui**: consente all'utente di effettuare le proprie richieste e di visualizzare sia le richieste non ancora servite, sia le operazioni effettuate.
- **FtpAgent**: legge le richieste di un utente ed effettua il download dei file ad esse relativi.

Gli agenti utilizzano per lo scambio di informazioni opportuni centri di tuple. Poiché l'uso di un server FTP richiede normalmente che l'utente sia identificato tramite account vengono creati tre centri di tuple per ogni nome utente registrato su un determinato server. I nomi dei centri di tuple sono quindi: `username_host_requests`, `username_host_downloads`, `username_host_errors`.

Potrebbe infatti verificarsi il caso in cui due utenti distinti hanno lo stesso nome registrato su server differenti e si avrebbe quindi sovrapposizione.

6.3.1 L'agente FtpGui

L'agente FtpGui presenta un'interfaccia grafica (fig. 6.3) tramite la quale è possibile effettuare richieste di download di file. E' necessario specificare il proprio nome di utente, e l'host sul quale risiede il server FTP. L'host può essere indicato sia tramite nome sia tramite indirizzo IP. Se ad esempio il server risiede sulla macchina locale è possibile inserire nel campo Host sia "localhost", sia "127.0.0.1".
Devono inoltre essere inseriti il nome assoluto del file locale e il nome del file remoto.

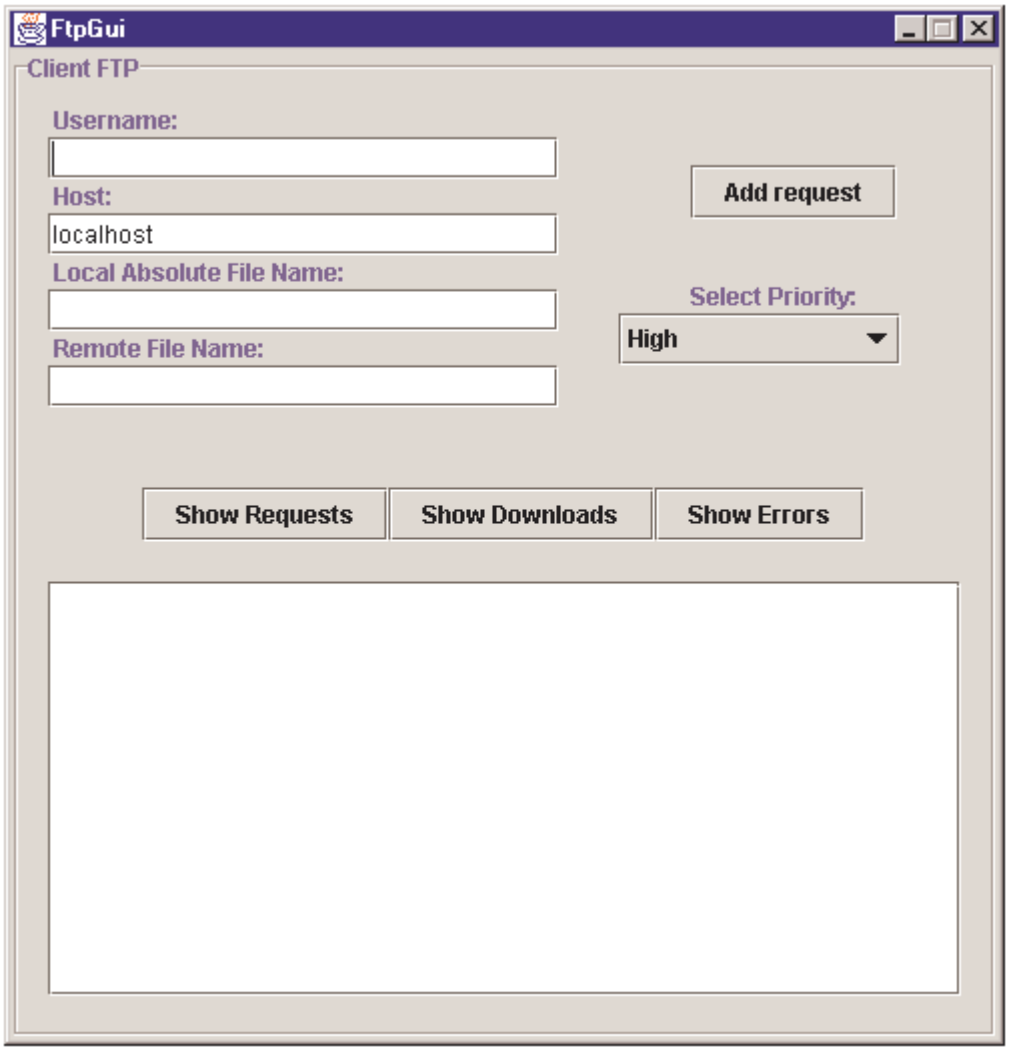


Figura 6.3 Interfaccia grafica dell'agente FtpGui

L’agente inserisce quindi nel centro di tuple `username_host_requests`, tuple nella forma:

```
request(localAbsoluteName,remoteName,priority,requestTime)
```

Quando si richiede di visualizzare l’insieme delle richieste o le operazioni effettuate o fallite, l’agente le legge dal corrispondente centro di tuple e le visualizza nell’area di testo.

6.3.2 L’agente FtpAgent

L’agente FtpAgent presenta un’interfaccia grafica (fig. 6.4) in cui l’utente deve inserire tutte le informazioni necessarie per la connessione al server.

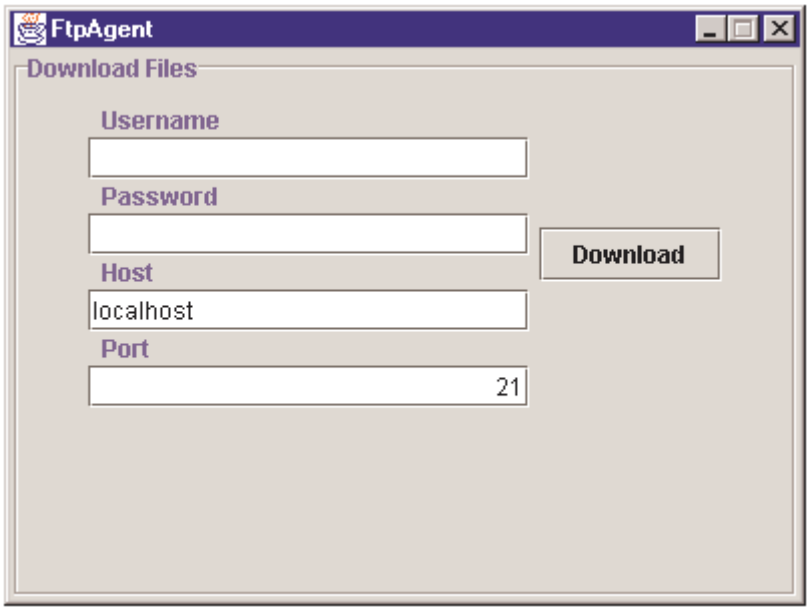


Figura 6.4 Interfaccia grafica dell'agente FtpAgent

L’host su cui risiede il server FTP può essere identificato sia tramite nome, sia tramite indirizzo IP. Deve anche essere specificata la porta sulla quale è disponibile il servizio, tipicamente 21. Infine nome utente e password devono identificare un utente registrato sul server.

A questo punto l’agente comincia ad effettuare le operazioni di download dei file, leggendo una richiesta dal centro di tuple `username_host_requests`, scegliendola tra quelle con priorità più alta. Cerca quindi di servire la richiesta e se l’operazione ha successo inserisce nel centro di tuple `username_host_downloads`, una tupla nella forma:

```
download(from,to,startTime,finishedTime)
```

Se invece l'operazione fallisce inserisce nel centro di tuple identificato da username_errors, una tupla nella forma:

```
error(from,to,startTime,errorMsg)
```

Quando non sono presenti richieste da servire, l'agente rimane in attesa.

6.3.3 Utilizzo di un componente preesistente

Java mette a disposizione nel package java.net classi utili all'implementazione del protocollo HTTP. Non fornisce invece ancora classi che implementino il protocollo FTP. Si è quindi ritenuto opportuno, ai fini dello sviluppo del progetto, utilizzare un componente già esistente: FTP Client Library realizzato dalla Enterprise Distributed Technologies.
Nel package *com.enterprisedt.net.ftp* sono contenute classi che implementano il protocollo FTP conformemente alle specifiche previste dal documento RFC 959.
In particolare la classe *FTPClient* fornisce un insieme di metodi che implementano i comandi FTP più usati. Inoltre, sono state utilizzate la classe *FTPReply*, che incapsula le risposte inviate dal server, e la classe *FTPTransferType* che consente di impostare il tipo dei dati trasferiti.

7 Collaudo del sistema

Il collaudo del sistema comporta la necessità di simularne l'architettura. E' quindi necessario configurare l'infrastruttura TuCSoN e disporre di server HTTP ed FTP standard su cui poter agire.
Per testare il componente FTP si è scelto di usare il server Meteor FTP, installandolo sulla stessa macchina su cui vengono eseguiti i componenti. Per quanto riguarda il componente HTTP invece, poiché i server che supportano questo protocollo consentono accessi senza la necessità di disporre di un account, il test avviene appoggiandosi direttamente ad un qualsiasi server HTTP di rete.

7.1 Configurazione del server FTP

Si è scelto di utilizzare Meteor come server FTP poiché esso supporta completamente le specifiche di protocollo contenute nel documento RFC 959.
Una volta installata ed eseguita l'applicazione è necessaria una fase di configurazione del server. Nella finestra mostrata in figura 7.1 occorre specificare la directory, in questo caso locale, a cui accedono gli utenti per trasferire i file.

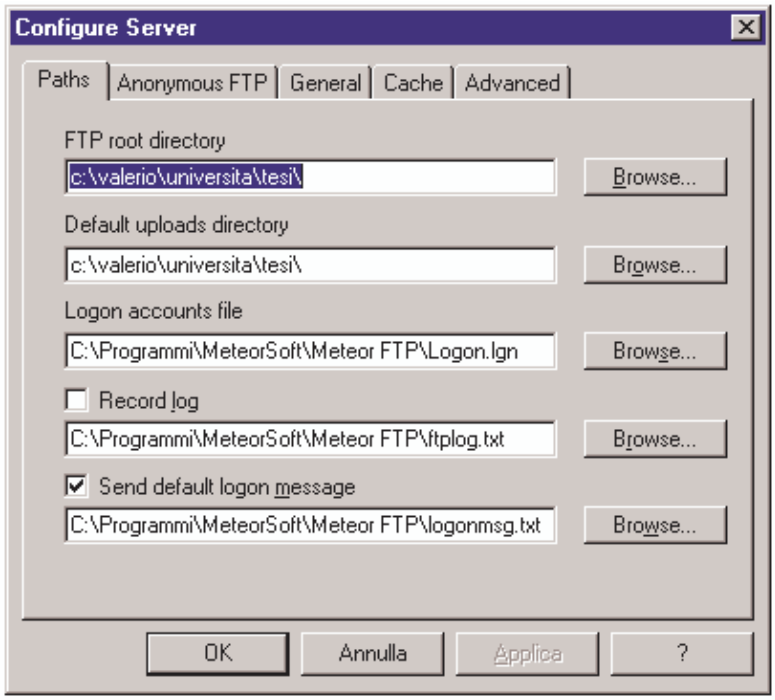


Figura 7.1 Interfaccia di configurazione di Meteor FTP

Si ha inoltre la possibilità di consentire accessi anonimi e di impostare altre opzioni, come ad esempio il numero massimo di connessioni.

A questo punto è necessario registrare un nuovo utente (fig. 7.2), inserendone nome e password e selezionane i diritti.

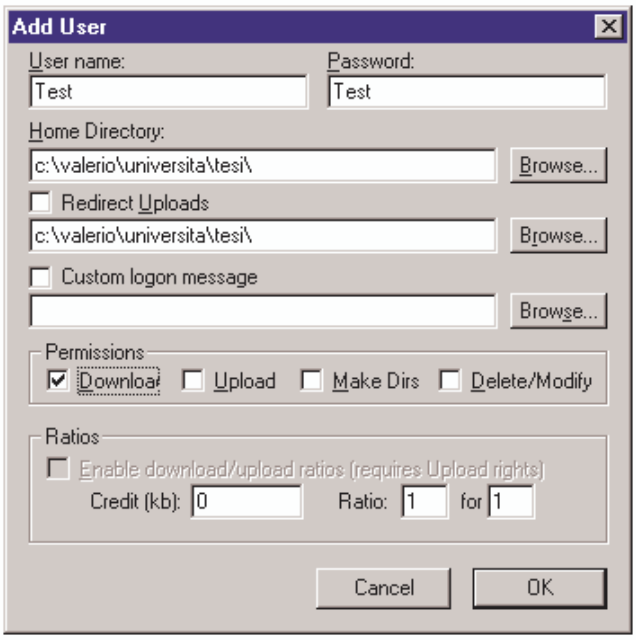


Figura 7.2 Finestra di configurazione utente

Infine, affinché sia possibile connettersi al server per effettuare le operazioni di test di download di file è necessario selezionare, dal menu *File*, l'opzione *Run in background*.

7.2 Configurazione dell'infrastruttura TuCSoN

L'attivazione dell'infrastruttura richiede l'installazione di almeno un nodo TuCSoN. E' possibile fare ciò o eseguendo il comando

```
java -cp dir/tucson.jar alice.tucson.runtime.Node
```

dove `dir` identifica la directory di installazione di tucson, o eseguendo il file `dir/bin/Node.bat`.
Se l'operazione è stata eseguita correttamente appare la seguente finestra:

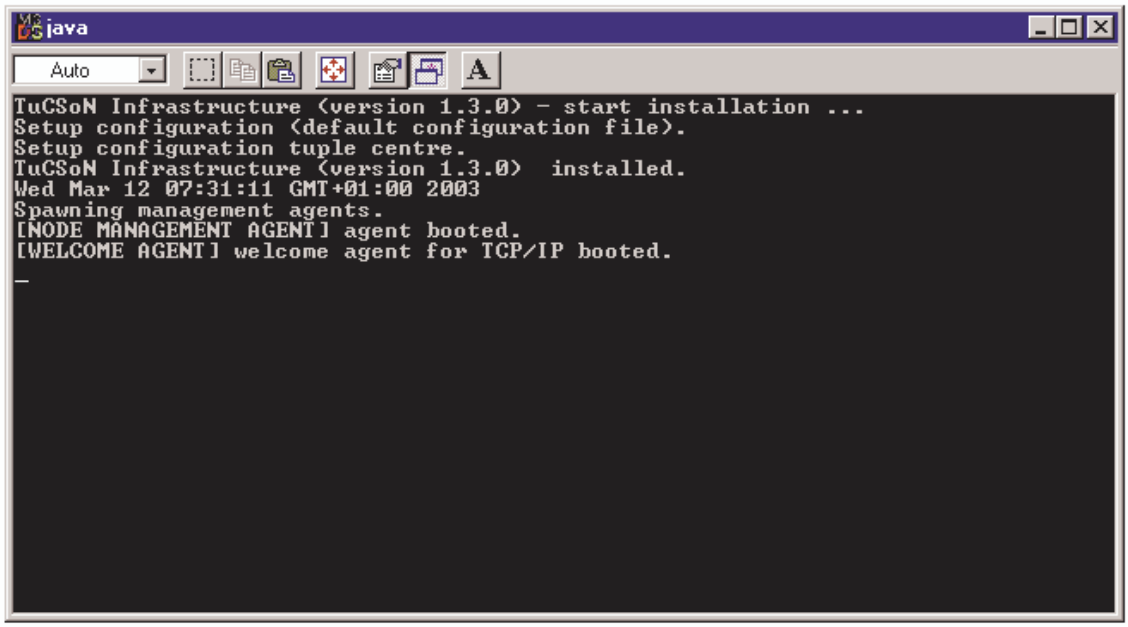


Figura 7.3 Finestra nodo TuCSoN

7.3 Test del componente HTTP

Per l'attivazione degli agenti è stato predisposto un pacchetto applicativo in cui sono contenuti i file batch per l'attivazione degli agenti. Una volta che il nodo TuCSoN è attivo, è quindi necessario eseguire i file `HttpAgent.bat` e `HttpGui.bat`.

Per effettuare il test si sceglie di richiedere lo scaricamento della home page del sito Internet del D.E.I.S. (Dipartimento Elettronica Informatica e Sistemistica) dell'Univerisità di Bologna. Una volta inserita la richiesta (fig. 7.4)

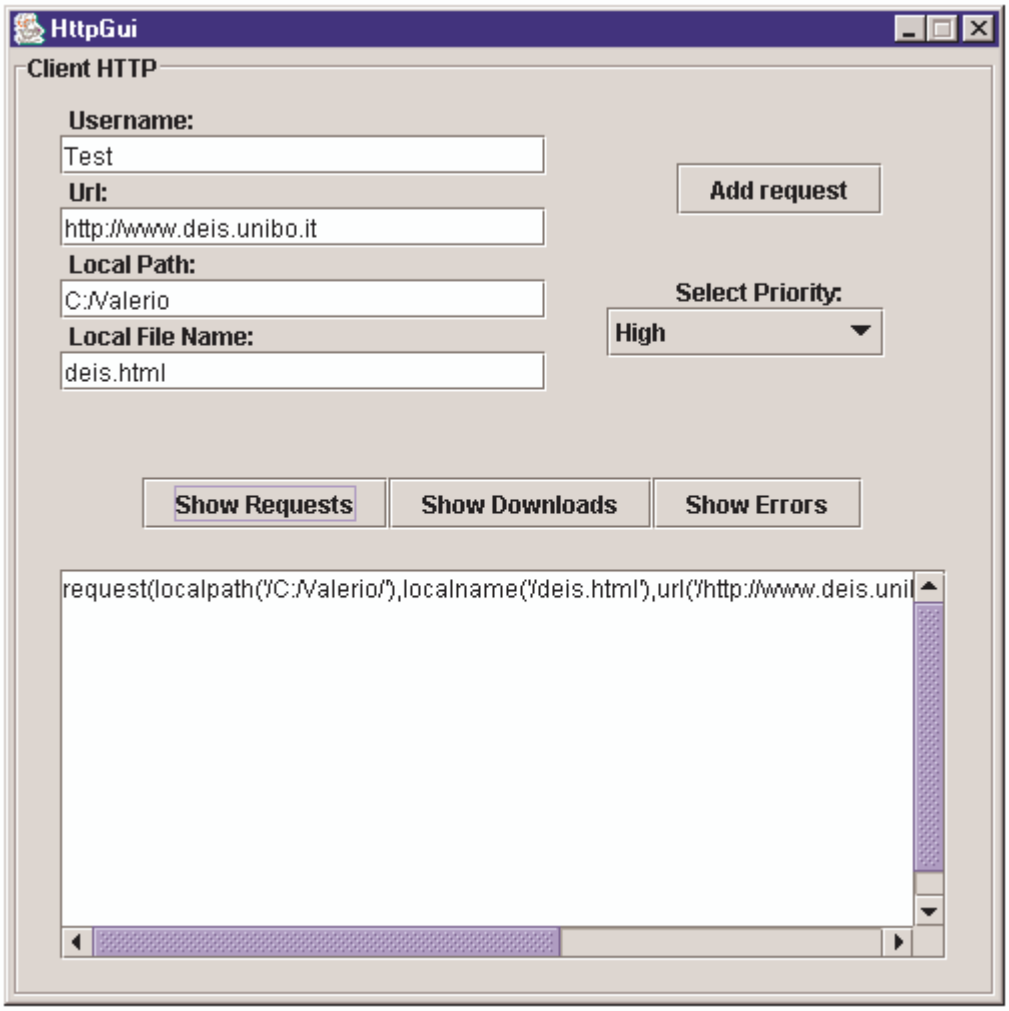


Figura 7.4 Test agente HttpGui

è possibile attivare l'agente HttpAgent ed effettuare il download del file (fig.7.5).

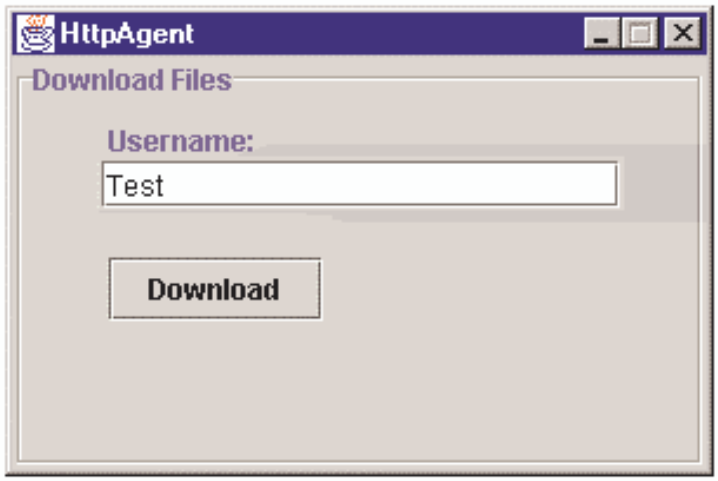


Figura 7.5 Test agente HttpAgent

Se l'operazione è stata completata con successo il centro di tuple `test_downloads` dovrebbe contenere una tupla. Si utilizza allora INSPECTOR di TuCSoN che consente di visualizzare lo stato della comunicazione e il comportamento di un centro di tuple. Visualizzando il contenuto di `test_downloads` si verifica che esso contiene una tupla relativa all'operazione di scaricamento effettuata (fig. 7.6).

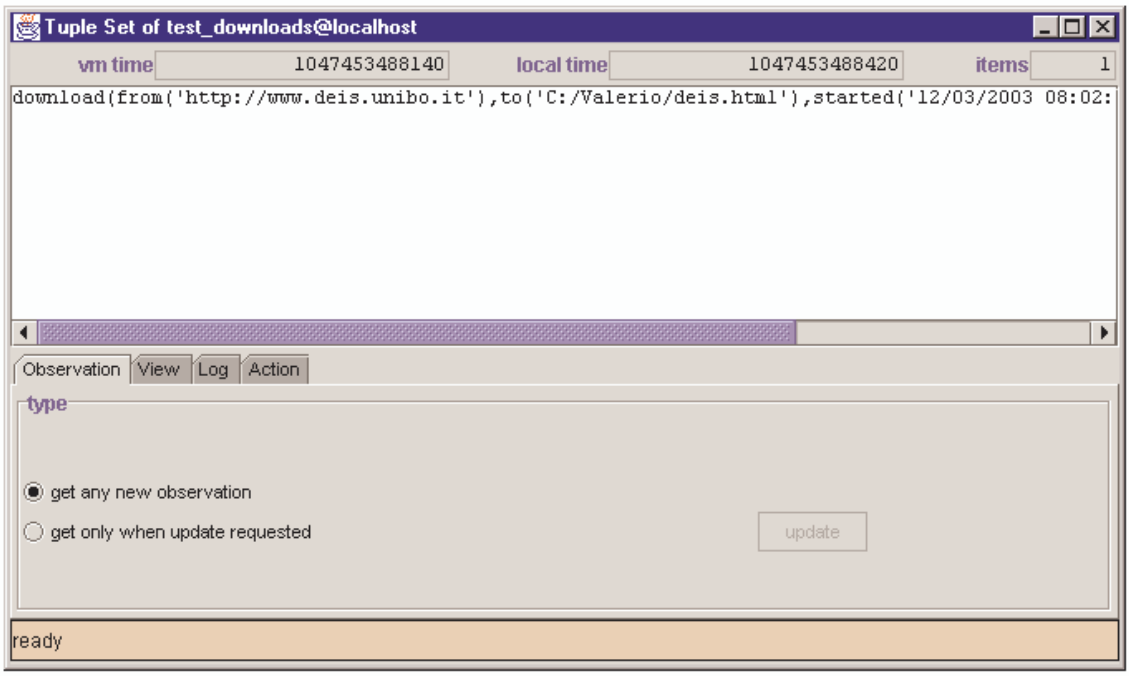


Figura 7.6 TuCSoN inspector

7.4 Test del componente FTP

Per l'attivazione degli agenti è stato predisposto un pacchetto applicativo in cui sono contenuti i file batch per l'attivazione degli agenti. Una volta che il nodo TuCSoN è attivo e che il server Meteor FTP è in attesa di richieste, è quindi necessario eseguire i file `FtpAgent.bat` e `FtpGui.bat`.

Sul server è stato registrato un utente con nome *Test* e password *Test*. Nell'effettuare la richiesta e nell'attivare il download sarà quindi necessario fare riferimento a questo account. Inoltre nel campo host dovrà sempre essere indicato *localhost*.

Per effettuare il test si inserisce la richiesta di un file contenuto nella directory del server. Una volta inserita la richiesta (fig. 7.7)

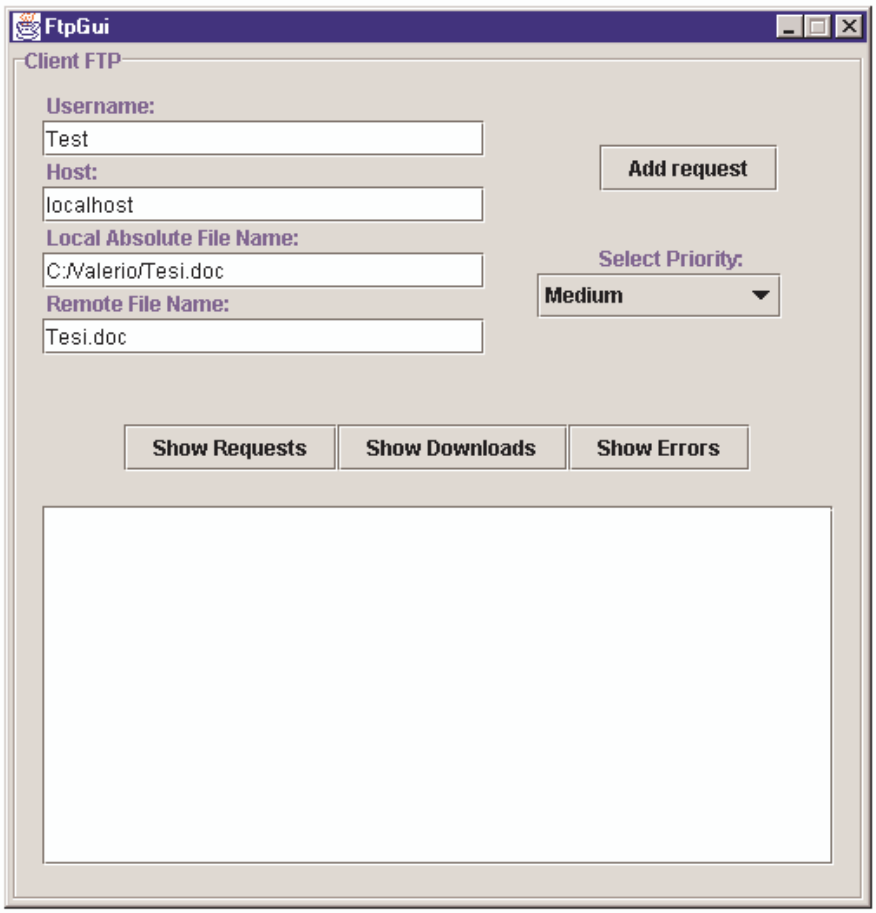


Figura 7.7 Test agente FtpGui

è possibile attivare l'agente FtpAgent ed effettuare il download del file (fig.7.8).

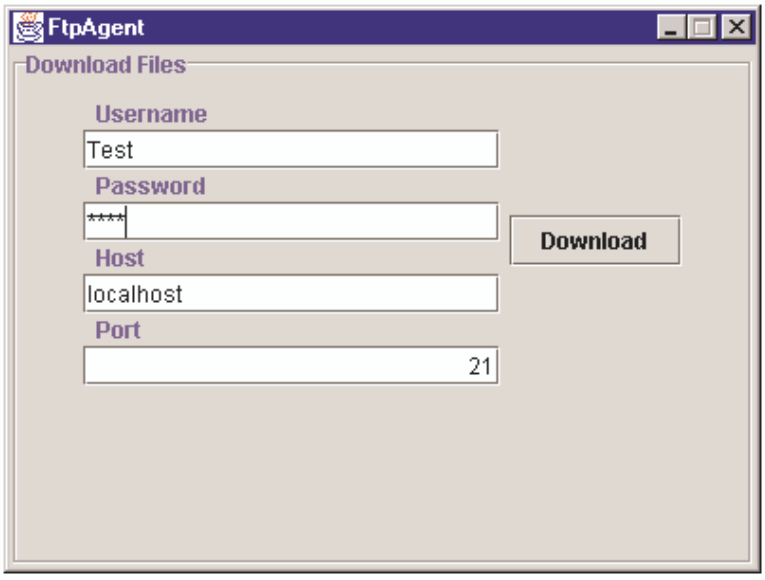


Figura 7.8 Test agente FtpAgent

Il server mostra la sequenza di comandi eseguiti e risposte inviate e ci consente di verificare che lo scaricamento del file è stato effettuato (fig. 7.9).

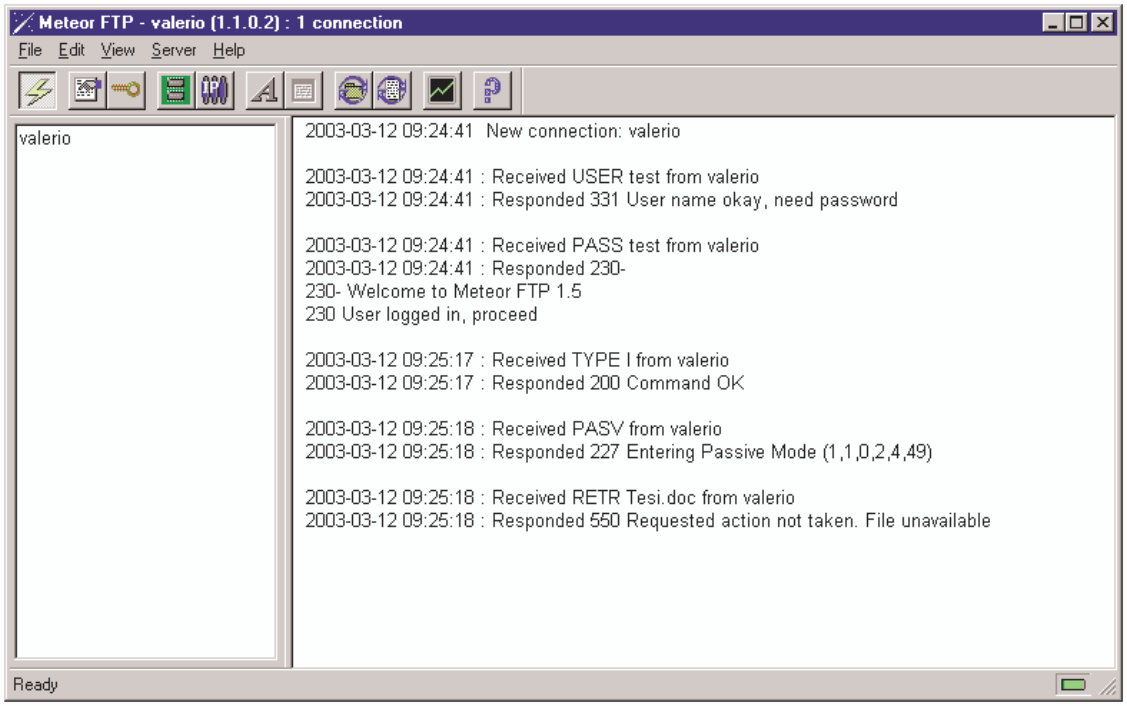


Figura 7.9 Server Meteor FTP

8 Conclusione

Nello sviluppo di questo progetto, è stata studiata la possibilità di estendere la tradizionale architettura client-server su cui si appoggiano i principali protocolli per il trasferimento di file, al fine di realizzare un sistema intelligente che fornisca servizi avanzati senza modificare i protocolli esistenti.

L'impiego dell'infrastruttura di coordinazione TuCSoN ha consentito l'inserimento di un livello intermedio nell'architettura del sistema. In particolare l'astrazione di coordinazione programmabile, costituita dal centro di tuple, ha svolto il ruolo di un'entità in grado di elaborare autonomamente le informazioni contenute, consentendo così la realizzazione di politiche complesse.

Il sistema sviluppato consente di effettuare lo scaricamento di file da un qualsiasi server standard HTTP o FTP. In particolare è stata sfruttata la possibilità di depositare informazioni relative alle operazioni desiderate nei centri di tuple in modo che i trasferimenti possano avvenire in un tempo differito e in un ordine deciso dall'utente in base all'assegnazione di priorità differenti alle richieste.

Altre politiche interessanti che potrebbero essere sviluppate sono, ad esempio, la possibilità di scelta da parte dell'utente di scaricare solo file di determinati formati oppure l'inserimento di timeout nelle operazioni di download.

Va inoltre notato come l'uso di tuple logiche per la rappresentazione di informazioni abbia portato alla realizzazione della stessa struttura per i differenti servizi di scaricamento di file tramite FTP ed HTTP. Si prospetta quindi la possibilità di ampliare il sistema attuale in modo che possa fornire

una visione unificata delle informazioni ottenute da servizi diversi (mail, FTP, HTTP, etc.).

Bibliografia

http://www.lia.deis.unibo.it/Research/TuCSoN

http://www.w3c.org

http://java.sun.com

RFC 2616 Hypertext Transfer Protocol – HTTP/1.1

RFC 959 File Transfer Protocol – FTP

Appendice

Componente HTTP

HttpAgent

```
import alice.tucson.api.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import modellohttp.*;

class HttpAgentPanel extends JPanel implements ActionListener{

  JLabel l1;
  JTextField username;
  JButton downloadButton;

  public HttpAgentPanel(){

    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(etched, "Download
Files");
    this.setBorder(titled);;

    Box b1 =Box.createVerticalBox();
    add(b1);

    l1 = new JLabel("Username:");
    username = new JTextField(20);
    username.requestFocus();

    Component cV = Box.createVerticalStrut(20);

    downloadButton = new JButton("Download");
    downloadButton.addActionListener(this);

    b1.add(l1);
    b1.add(username);
    b1.add(cV);
    b1.add(downloadButton);
  }

  public void actionPerformed(ActionEvent e){
    try{
      AgentId aid=new AgentId("download");
      new DownloadAgent(aid, username.getText().trim()).spawn();
    }
    catch(Exception ex){
      ex.printStackTrace();
    }
  }
```

```
}

public class HttpAgent {

   public static void main(String[] args) {
   try{

      JFrame frame = new JFrame("HttpAgent");
      frame.setBounds(50, 50, 300, 200);
      frame.setResizable(false);
      Container contentPane = frame.getContentPane();
      HttpAgentPanel p = new HttpAgentPanel();
      contentPane.add(p);
      frame.show();

      frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      });
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

HttpGui

```
import alice.tucson.api.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.net.*;
import java.util.*;
import modellohttp.*;

class HttpGuiPanel extends JPanel implements ActionListener{

  JLabel l1, l2, l3, l4, l5;
  JTextField username, url, localPath, localName;
  JComboBox priority;
  JButton requestButton, showRequestsButton, showDownloadsButton,
showErrorsButton;
  JTextArea area;
  JScrollPane scrollPane;

  public HttpGuiPanel(){

    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(etched, "Client
HTTP");
    this.setBorder(titled);;

    Box b1 = Box.createVerticalBox();
    Box b2 = Box.createHorizontalBox();
    Box b3 = Box.createHorizontalBox();
    Box b4 = Box.createVerticalBox();
    Box b5 = Box.createVerticalBox();

    add(b1);
    add(b2);
    add(b3);
    add(b4);
    add(b5);

    l1 = new JLabel("Username:");
    username = new JTextField(20);
    username.requestFocus();

    l2 = new JLabel("Url:");
    url = new JTextField("http://", 20);

    l3 = new JLabel("Local Path:");
    localPath = new JTextField(20);

    l4 = new JLabel("Local File Name:");
    localName = new JTextField(20);

    l5 = new JLabel("Select Priority:");
    priority = new JComboBox();
    priority.setEditable(false);
    priority.addItem("High");
    priority.addItem("Medium");
```

```
  priority.addItem("Low");

  Component cV1 = Box.createVerticalStrut(30);
  Component cV2 = Box.createVerticalStrut(30);
  Component cV3 = Box.createVerticalStrut(40);
  Component cV4 = Box.createVerticalStrut(20);
  Component cV5 = Box.createVerticalStrut(20);
  Component cH1 = Box.createHorizontalStrut(30);
  Component cH2 = Box.createHorizontalStrut(30);

  requestButton = new JButton("Add request");
  requestButton.addActionListener(this);

  showRequestsButton = new JButton("Show Requests");
  showRequestsButton.addActionListener(this);

  showDownloadsButton = new JButton("Show Downloads");
  showDownloadsButton.addActionListener(this);

  showErrorsButton = new JButton("Show Errors");
  showErrorsButton.addActionListener(this);

  area = new JTextArea(12,40);
  area.setEditable(false);
  scrollPane = new JScrollPane(area);

  b1.add(l1);
  b1.add(username);
  b1.add(l2);
  b1.add(url);
  b1.add(l3);
  b1.add(localPath);
  b1.add(l4);
  b1.add(localName);
  b5.add(cV1);
  b5.add(requestButton);
  b5.add(cV2);
  b5.add(l5);
  b5.add(priority);
  b5.add(cV5);
  b2.add(b1);
  b2.add(cH1);
  b2.add(b5);
  b2.add(cH2);
  b3.add(showRequestsButton);
  b3.add(showDownloadsButton);
  b3.add(showErrorsButton);
  b4.add(b2);
  b4.add(cV3);
  b4.add(b3);
  b4.add(cV4);
  b4.add(scrollPane);
}

public void actionPerformed(ActionEvent e){

  Object choose=e.getSource();

  if (choose==requestButton){
    try{
```

```
        AgentId aid=new AgentId("addrequest");
        new AddRequestAgent(aid, username.getText().trim(),
url.getText().trim(), localPath.getText().trim(),
localName.getText().trim(),
(String)priority.getSelectedItem()).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
      url.setText("");
      localPath.setText("");
      localName.setText("");
      url.requestFocus();
    }

    if (choose==showRequestsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showr");
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_requests");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
    }

    if (choose==showDownloadsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showd");
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_downloads");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
    }

    if (choose==showErrorsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showe");
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_errors");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
    }
  }
}

public class HttpGui {

  public static void main(String[] args) {
    try{
```

```
      JFrame frame = new JFrame("HttpGui");
      frame.setBounds(500, 50, 500, 500);
      frame.setResizable(false);
      Container contentPane = frame.getContentPane();
      HttpGuiPanel p = new HttpGuiPanel();
      contentPane.add(p);
      frame.show();

      frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      });
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Package modellohttp

AddRequestAgent

```
package modellohttp;

import alice.tucson.api.*;
import alice.logictuple.*;
import java.net.*;
import java.util.*;

public class AddRequestAgent extends Agent{

  private String user="";
  private String url="";
  private String localPath="";
  private String localName="";
  private String priority="";

  public AddRequestAgent(AgentId a, String usr, String u, String lp,
String ln, String p)throws TucsonException{
    super(a);
    user=usr;
    url=u;
    if (!(lp.endsWith("/")||lp.endsWith("\\")))lp=lp.concat("/");
    localPath=lp;
    if (!(ln.endsWith(".html")))ln=ln.concat(".html");
    localName=ln;
    priority=p;
  }

  protected void body(){
    try{
      TupleCentreId tid = new
TupleCentreId(user.toLowerCase()+"_requests");
      System.out.println(tid.getName());

      //Si creano ora e data della richiesta
      Calendar calendar = new GregorianCalendar();
      String timeReq = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));
      out(tid, new LogicTuple("request", new Value("localpath", new
Value("/"+localPath)), new Value("localname", new
Value("/"+localName)), new Value("url", new Value("/"+url)), new
```

```
Value("priority", new Value(priority)), new Value("requested", new
Value(timeReq))));
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

AgentReader

```
package modellohttp;

import alice.tucson.api.*;
import alice.logictuple.*;
import alice.tuprolog.Term;
import alice.tuprolog.Struct;
import javax.swing.*;
import java.net.*;
import java.io.*;

public class AgentReader extends Agent{

  TupleCentreId tid;
  String tupla="";
  LogicTuple tupleTemplate=null;
  LogicTuple tuple=null;
  JTextArea area;
  alice.tuprolog.Var head, tail;

  public AgentReader(AgentId a, TupleCentreId t, JTextArea area)
throws TucsonException{
    super(a);
    tid = new TupleCentreId(t.getName());
    this.area=area;
  }

  protected void body(){
    try {
      try {
        InputStream is =
ClassLoader.getSystemResourceAsStream("rdall.rsp");
        BufferedInputStream in = new BufferedInputStream(is);
        String text = alice.util.Tools.loadText(in);
        setSpec(tid,text);
      }
      catch (Exception e){
        e.printStackTrace();
      }
      Var vt=new Var("Lista");
      TupleTemplate t = new TupleTemplate(tid);
      tupleTemplate = t.createTupleTemplate();
      LogicTuple tlist = in(tid,new LogicTuple("rdall",new
TupleArgument(tupleTemplate.toTerm()),vt));

      Term term = vt.toTerm();
      Struct lista = (Struct) term.getTerm();

      while(!lista.unify(new Struct())){
        head = new alice.tuprolog.Var("Head");
        tail = new alice.tuprolog.Var("Tail");
        boolean res = lista.unify(new Struct(head, tail));
        lista = (Struct) tail.getTerm();

        tupla = head.toString();
        tuple = LogicTuple.parse(tupla);
        tupla = tuple.getArg(1).toString();
```

```
          area.append(tupla+"\n");
        }
      }
      catch (Exception e){
        e.printStackTrace();
      }
    }
  }
```

DownloadAgent

```
package modellohttp;

import alice.tucson.api.*;
import alice.logictuple.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class DownloadAgent extends Agent{

  private String user="";
  private Calendar calendar=null;
  public DownloadAgent(AgentId a, String usr) throws TucsonException{
    super(a);
    user=usr;
  }

  protected void body(){
    try{

      TupleCentreId tidReq = new
TupleCentreId(user.toLowerCase()+"_requests");
      LogicTuple request=null;
      String localPath="", localName="", sUrl="", timeStart="",
timeFinish="";
      String inline="", sHtml="", img="", src="", file="",
sUrlBase="", imgName = "";
      char separator = '/';
      FileWriter html = null;
      File f = null;
      URL pageUrl = null, imgUrl = null;
      Pattern pImg = Pattern.compile("<[Ii][Mm][Gg][^>]*>");
      Pattern pSrc = Pattern.compile("[Ss][Rr][Cc][ ]*=[
]*[\"][^\"]*[\"]");
      Pattern pFile = Pattern.compile("[\"][^\"]*[\"]");
      Pattern pBase = Pattern.compile("http://[^/]*");
      Matcher mImg = null, mSrc = null, mFile = null, mBase = null;

      while(true){
        try{
          rd(tidReq, new LogicTuple("request", new Var(), new Var(),
new Var(), new Var(), new Var()));
          request = inp(tidReq, new LogicTuple("request", new Var(),
new Var(), new Var(), new Value("priority", new Value("High")), new
Var()));
          if (request==null)request = inp(tidReq, new
LogicTuple("request", new Var(), new Var(), new Var(), new
Value("priority", new Value("Medium")), new Var()));
          if (request==null)request = inp(tidReq, new
LogicTuple("request", new Var(), new Var(), new Var(), new
Value("priority", new Value("Low")), new Var()));
          if (request!=null){
            localPath = request.getArg(0).getArg(0).toString();
            localName = request.getArg(1).getArg(0).toString();
            sUrl = request.getArg(2).getArg(0).toString();
```

```
            // Si ripulisce la stringa
            localPath = localPath.substring(2, localPath.length()-1);
            localName = localName.substring(2, localName.length()-1);
            sUrl = sUrl.substring(2, sUrl.length()-1);
            System.out.println(localPath);
            System.out.println(localName);
            System.out.println(sUrl);

            //inizia il download dei file
            calendar = new GregorianCalendar();
            timeStart = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));
            System.out.println(timeStart);
            String subUrl = sUrl.substring(7,sUrl.length());
            InetAddress net = InetAddress.getByName(subUrl);
            System.out.println("Download started at:"+ timeStart);
            pageUrl = new URL(sUrl);

            //si crea la cartella per le immagini associate alla
pagina
            String fold= new String(localName.substring(0,
localName.length()-5)+"_files");
            f=new File(localPath+fold);
            f.mkdir();
            mBase=pBase.matcher(sUrl);
            if(mBase.find())sUrlBase=mBase.group();

            sHtml="";
            BufferedReader inHtml = new BufferedReader(new
InputStreamReader(pageUrl.openStream()));
            while((inline=inHtml.readLine())!=null){
              sHtml+=inline;
            }
            inHtml.close();
            mImg = pImg.matcher(sHtml);
            while (mImg.find()){
              img = mImg.group();
              mSrc = pSrc.matcher(img);
              if (mSrc.find()){
                src=mSrc.group();
                mFile = pFile.matcher(src);
                if (mFile.find()){
                  file=mFile.group();
                  file = file.substring(1,file.length()-1);
                  if(!file.startsWith("http:")){
                    System.out.println("Downloading.." + file);
                    char[] fileChar = file.toCharArray();
                    for(int i=0; i<fileChar.length; i++){
```

```
                    if (fileChar[i]==separator) {
                      imgName = "";
                    }
                    imgName += fileChar[i];
                  }

                  Pattern pNSrc = Pattern.compile(file);
                  Matcher mNSrc = pNSrc.matcher(sHtml);
                  sHtml = mNSrc.replaceFirst(fold+imgName);

                  if (file.startsWith("..")){
                    file=file.substring(2,file.length());
                    imgUrl = new URL(sUrlBase+file);
                  }
                  if(file.startsWith("/"))imgUrl = new
URL(sUrlBase+file);
                  else imgUrl = new URL(sUrlBase+separator+file);
                  f = new File(localPath+fold+imgName);

System.out.println("Created:"+localPath+fold+imgName);
                  int b=0;
                  FileOutputStream fout = new FileOutputStream(f);
                  BufferedInputStream inFile = new
BufferedInputStream(imgUrl.openStream());
                  while((b=inFile.read())>=0){
                    fout.write(b);
                  }
                  inFile.close();
                }
              }
            }
          }
          html = new FileWriter(localPath+localName);
          html.write(sHtml, 0, sHtml.length());
          html.flush();
          html.close();

          //download dei file terminato
          alendar = new GregorianCalendar();
          timeFinish = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));
          System.out.println("Download finished at:" + timeFinish);
          TupleCentreId tidDl = new
TupleCentreId(user.toLowerCase()+"_downloads");
          out(tidDl, new LogicTuple("download", new Value("from",
new Value(sUrl)), new Value("to", new Value(localPath+localName)), new
Value("started", new Value(timeStart)), new Value("finished", new
Value(timeFinish))));
        }
```

```
        }
        catch (Exception e){
          String error = e.toString();
          System.out.println("Download failed:" + error);
          TupleCentreId tidErr = new
TupleCentreId(user.toLowerCase()+"_errors");
          out(tidErr, new LogicTuple("error", new Value("from", new
Value(sUrl)), new Value("to", new Value(localPath+localName)), new
Value("started", new Value(timeStart)), new Value("errormsg", new
Value(error))));
        }
      }
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

TupleTemplate

```
package modellohttp;

import alice.tucson.api.*;
import alice.logictuple.*;

public class TupleTemplate {
  TupleCentreId tid;
  LogicTuple t=null;

  public TupleTemplate(TupleCentreId t) throws TucsonException{
    tid = new TupleCentreId(t.getName());
  }

  private boolean strInStr(String s1, String s2){
    int n = s2.length();
    do {
      if(s1.equals(s2)) return true;
      s1=s1.substring(1,s1.length());
    } while(s1.length()>=n);
    return false;
  }

  public LogicTuple createTupleTemplate(){
    if (strInStr(tid.getName(), "requests")){
      t = new LogicTuple("request", new Var(), new Var(), new Var(),
new Var(), new Var());
    }
    if (strInStr(tid.getName(), "downloads")){
      t = new LogicTuple("download", new Var(), new Var(), new Var(),
new Var());
    }
    if (strInStr(tid.getName(), "errors")){
      t = new LogicTuple("error", new Var(), new Var(), new Var(), new
Var());
    }
    return t;
  }

}
```

Componente FTP

FtpAgent

```
import alice.tucson.api.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import modelloftp.*;

class FtpAgentPanel extends JPanel implements ActionListener{

  JLabel l1, l2, l3, l4;
  JTextField username, host, port;
  JPasswordField password;
  JButton downloadButton;

  public FtpAgentPanel(){

    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(etched, "Download
Files");
    this.setBorder(titled);;

    Box b1 =Box.createVerticalBox();
    Box b2 = Box.createHorizontalBox();
    add(b1);
    add(b2);

    l1 = new JLabel("Username");
    username = new JTextField(20);
    username.requestFocus();

    l2 = new JLabel("Password");
    password = new JPasswordField(20);

    l3 = new JLabel("Host");
    host = new JTextField("localhost", 20);

    l4 = new JLabel("Port");
    port = new JTextField("21", 20);
    port.setHorizontalAlignment(JTextField.RIGHT);

    downloadButton = new JButton("Download");
    downloadButton.addActionListener(this);

    b1.add(l1);
    b1.add(username);
    b1.add(l2);
    b1.add(password);
    b1.add(l3);
    b1.add(host);
    b1.add(l4);
```

```
    b1.add(port);
    b2.add(downloadButton);

  }

  public void actionPerformed(ActionEvent e){
    try{
      int p = Integer.parseInt(port.getText().trim());
      AgentId aid=new AgentId("download");
      new DownloadAgent(aid, host.getText().trim(), p,
username.getText().trim(), new
String(password.getPassword())).spawn();
    }
    catch(Exception ex){
      ex.printStackTrace();
    }
  }
}

public class FtpAgent {

   public static void main(String[] args) {
   try{

      JFrame frame = new JFrame("FtpAgent");
      frame.setBounds(50, 50, 400, 300);
      frame.setResizable(false);
      Container contentPane = frame.getContentPane();
      FtpAgentPanel p = new FtpAgentPanel();
      contentPane.add(p);
      frame.show();

      frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      });
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

FtpGui

```
import alice.tucson.api.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.net.*;
import java.util.*;
import modelloftp.*;

class FtpGuiPanel extends JPanel implements ActionListener{

  JLabel l1, l2, l3, l4, l5;
  JTextField username, host, localPath, remotePath;
  JComboBox priority;
  JButton requestButton, showRequestsButton, showDownloadsButton,
showErrorsButton;
  JTextArea area;
  JScrollPane scrollPane;

  public FtpGuiPanel(){

    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(etched, "Client
FTP");
    this.setBorder(titled);;

    Box b1 = Box.createVerticalBox();
    Box b2 = Box.createHorizontalBox();
    Box b3 = Box.createHorizontalBox();
    Box b4 = Box.createVerticalBox();
    Box b5 = Box.createVerticalBox();
    add(b1);
    add(b2);
    add(b3);
    add(b4);
    add(b5);

    l1 = new JLabel("Username:");
    username = new JTextField(20);
    username.requestFocus();

    l2 = new JLabel("Host:");
    host = new JTextField("localhost", 20);

    l3 = new JLabel("Local Absolute File Name:");
    localPath = new JTextField(20);

    l4 = new JLabel("Remote File Name:");
    remotePath = new JTextField(20);

    l5 = new JLabel("Select Priority:");
    priority = new JComboBox();
    priority.setEditable(false);
    priority.addItem("High");
    priority.addItem("Medium");
```

```
  priority.addItem("Low");

  Component cV1 = Box.createVerticalStrut(30);
  Component cV2 = Box.createVerticalStrut(30);
  Component cV3 = Box.createVerticalStrut(40);
  Component cV4 = Box.createVerticalStrut(20);
  Component cV5 = Box.createVerticalStrut(20);
  Component cH1 = Box.createHorizontalStrut(30);
  Component cH2 = Box.createHorizontalStrut(30);

  requestButton = new JButton("Add request");
  requestButton.addActionListener(this);

  showRequestsButton = new JButton("Show Requests");
  showRequestsButton.addActionListener(this);

  showDownloadsButton = new JButton("Show Downloads");
  showDownloadsButton.addActionListener(this);

  showErrorsButton = new JButton("Show Errors");
  showErrorsButton.addActionListener(this);

  area = new JTextArea(12,40);
  area.setEditable(false);
  scrollPane = new JScrollPane(area);

  b1.add(l1);
  b1.add(username);
  b1.add(l2);
  b1.add(host);
  b1.add(l3);
  b1.add(localPath);
  b1.add(l4);
  b1.add(remotePath);
  b5.add(cV1);
  b5.add(requestButton);
  b5.add(cV2);
  b5.add(l5);
  b5.add(priority);
  b5.add(cV5);
  b2.add(b1);
  b2.add(cH1);
  b2.add(b5);
  b2.add(cH2);
  b3.add(showRequestsButton);
  b3.add(showDownloadsButton);
  b3.add(showErrorsButton);
  b4.add(b2);
  b4.add(cV3);
  b4.add(b3);
  b4.add(cV4);
  b4.add(scrollPane);
}

public void actionPerformed(ActionEvent e){

  Object choose=e.getSource();

  if (choose==requestButton){
    try{
```

```
        AgentId aid=new AgentId("addrequest");
        new AddRequestAgent(aid, username.getText().trim(),
host.getText().trim(), localPath.getText().trim(),
remotePath.getText().trim(),
(String)priority.getSelectedItem()).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
      localPath.setText("");
      remotePath.setText("");
      localPath.requestFocus();
    }

    if (choose==showRequestsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showr");
        InetAddress iNetHost = InetAddress.getByName(host.getText());
        String h = iNetHost.getHostAddress().replace('.','_');
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_"+h.toLowerCase()+"_r
equests");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
    }

    if (choose==showDownloadsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showd");
        InetAddress iNetHost = InetAddress.getByName(host.getText());
        String h = iNetHost.getHostAddress().replace('.','_');
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_"+h.toLowerCase()+"_d
ownloads");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
    }

    if (choose==showErrorsButton){
      try{
        area.setText("");
        AgentId aid=new AgentId("showe");
        InetAddress iNetHost = InetAddress.getByName(host.getText());
        String h = iNetHost.getHostAddress().replace('.','_');
        TupleCentreId tidReq = new
TupleCentreId(username.getText().toLowerCase()+"_"+h.toLowerCase()+"_e
rrors");
        new AgentReader(aid, tidReq, this.area).spawn();
      }
      catch(Exception ex){
        ex.printStackTrace();
      }
```

```
    }
  }
}

public class FtpGui {

  public static void main(String[] args) {
    try{
      JFrame frame = new JFrame("FtpGui");
      frame.setBounds(500, 50, 500, 500);
      frame.setResizable(false);
      Container contentPane = frame.getContentPane();
      FtpGuiPanel p = new FtpGuiPanel();
      contentPane.add(p);
      frame.show();

      frame.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
      });
    }
    catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

Package modelloftp

AddRequestAgent

```
package modelloftp;
import com.enterprisedt.net.ftp.*;
import alice.tucson.api.*;
import alice.logictuple.*;
import java.net.*;
import java.util.*;

public class AddRequestAgent extends Agent{

  private String user="";
  private String host="";
  private String localPath="";
  private String remotePath="";
  private String priority="";

  public AddRequestAgent(AgentId a, String usr, String h, String lp,
String rp, String p)throws TucsonException{
    super(a);
    user=usr;
    host=h;
    localPath=lp;
    remotePath=rp;
    priority=p;
  }

  protected void body(){
    try{
      InetAddress iNetHost = InetAddress.getByName(host);
      host = iNetHost.getHostAddress().replace('.','_');
      TupleCentreId tid = new
TupleCentreId(user.toLowerCase()+"_"+host.toLowerCase()+"_requests");

      //Si creano ora e data della richiesta
      Calendar calendar = new GregorianCalendar();
      String timeReq = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));
      out(tid, new LogicTuple("request", new Value("localpath", new
Value("/"+localPath)), new Value("remotepath", new
Value("/"+remotePath)), new Value("priority", new Value(priority)),
new Value("requested", new Value(timeReq))));
    }
```

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

AgentReader

```
package modelloftp;
import alice.tucson.api.*;
import alice.logictuple.*;
import alice.tuprolog.Term;
import alice.tuprolog.Struct;
import javax.swing.*;
import java.net.*;
import java.io.*;

public class AgentReader extends Agent{

  TupleCentreId tid;
  String tupla="";
  LogicTuple tupleTemplate=null;
  LogicTuple tuple=null;
  JTextArea area;
  alice.tuprolog.Var head, tail;

  public AgentReader(AgentId a, TupleCentreId t, JTextArea area)
throws TucsonException{
    super(a);
    tid = new TupleCentreId(t.getName());
    this.area=area;
  }

  protected void body(){
    try {
      try {
        InputStream is =
ClassLoader.getSystemResourceAsStream("rdall.rsp");
        BufferedInputStream in = new BufferedInputStream(is);
        String text = alice.util.Tools.loadText(in);
        setSpec(tid,text);
      }
      catch (Exception e){
        e.printStackTrace();
      }
      Var vt=new Var("Lista");
      TupleTemplate t = new TupleTemplate(tid);
      tupleTemplate = t.createTupleTemplate();
      LogicTuple tlist = in(tid,new LogicTuple("rdall",new
TupleArgument(tupleTemplate.toTerm()),vt));

      Term term = vt.toTerm();
      Struct lista = (Struct) term.getTerm();

      while(!lista.unify(new Struct())){
        head = new alice.tuprolog.Var("Head");
        tail = new alice.tuprolog.Var("Tail");
        boolean res = lista.unify(new Struct(head, tail));
        lista = (Struct) tail.getTerm();

        tupla = head.toString();
        tuple = LogicTuple.parse(tupla);
        tupla = tuple.getArg(1).toString();
        area.append(tupla+"\n");
```

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

DownloadAgent

```
package modelloftp;
import com.enterprisedt.net.ftp.*;
import alice.tucson.api.*;
import alice.logictuple.*;
import java.net.*;
import java.util.*;

public class DownloadAgent extends Agent{

  private String host="";
  private int port;
  private String user="";
  private String password="";
  private Calendar calendar=null;

  public DownloadAgent(AgentId a, String h, int p, String usr, String
pwd) throws TucsonException{
    super(a);
    host=h;
    port=p;
    user=usr;
    password=pwd;
  }

  protected void body(){
    try{
      InetAddress InetHost = InetAddress.getByName(host);
      FTPClient client = new FTPClient(InetHost, port);
      client.login(user, password);
      System.out.println(client.getLastValidReply().getReplyCode() +
client.getLastValidReply().getReplyText());

      String host = InetHost.getHostAddress().replace('.','_');
      TupleCentreId tidReq = new
TupleCentreId(user.toLowerCase()+"_"+host.toLowerCase()+"_requests");
      LogicTuple request=null;
      String localPath="", remotePath="", timeStart="", timeFinish="";

      while(true){
        try{
          rd(tidReq, new LogicTuple("request", new Var(), new Var(),
new Var(), new Var()));
          request = inp(tidReq, new LogicTuple("request", new Var(),
new Var(), new Value("priority", new Value("High")), new Var()));
          if (request==null)request = inp(tidReq, new
LogicTuple("request", new Var(), new Var(), new Value("priority", new
Value("Medium")), new Var()));
          if (request==null)request = inp(tidReq, new
LogicTuple("request", new Var(), new Var(), new Value("priority", new
Value("Low")), new Var()));

          if (request!=null){

            localPath = request.getArg(0).getArg(0).toString();
            remotePath = request.getArg(1).getArg(0).toString();
```

```
            // Si ripulisce la stringa
            localPath = localPath.substring(2, localPath.length()-1);
            remotePath = remotePath.substring(2, remotePath.length()-
1);
            client.setType(FTPTransferType.BINARY);
            //Inizia il download
            calendar = new GregorianCalendar();
            timeStart = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));
            client.get(localPath, remotePath);
            //Download terminato
            calendar = new GregorianCalendar();
            timeFinish = new
String(""+(calendar.get(Calendar.DAY_OF_MONTH)<10 ?
"0"+calendar.get(Calendar.DAY_OF_MONTH) :
""+calendar.get(Calendar.DAY_OF_MONTH))+"/"+((calendar.get(Calendar.MO
NTH)+1)<10 ? "0"+(calendar.get(Calendar.MONTH)+1) :
""+(calendar.get(Calendar.MONTH)+1))+"/"+calendar.get(Calendar.YEAR)+"
"+(calendar.get(Calendar.HOUR_OF_DAY)<10 ?
"0"+calendar.get(Calendar.HOUR_OF_DAY) :
""+calendar.get(Calendar.HOUR_OF_DAY))+":"+(calendar.get(Calendar.MINU
TE)<10 ? "0"+calendar.get(Calendar.MINUTE) :
""+calendar.get(Calendar.MINUTE))+":"+(calendar.get(Calendar.SECOND)<1
0 ? "0"+calendar.get(Calendar.SECOND) :
""+calendar.get(Calendar.SECOND)));

System.out.println(client.getLastValidReply().getReplyCode() +
client.getLastValidReply().getReplyText());
            TupleCentreId tidDl = new
TupleCentreId(user.toLowerCase()+"_"+host.toLowerCase()+"_downloads");
            out(tidDl, new LogicTuple("download", new Value("from",
new Value(remotePath)), new Value("to", new Value(localPath)), new
Value("started", new Value(timeStart)), new Value("finished", new
Value(timeFinish))));
          }
        }

        catch (Exception e){
          String error = e.toString();
          TupleCentreId tidErr = new
TupleCentreId(user.toLowerCase()+"_"+host.toLowerCase()+"_errors");
          out(tidErr, new LogicTuple("error", new Value("from", new
Value(remotePath)), new Value("to", new Value(localPath)), new
Value("started", new Value(timeStart)), new Value("errormsg", new
Value(error))));
        }

      }
    }
```

```
    catch (Exception e){
      e.printStackTrace();

    }
  }
}
```

TupleTemplate

```
package modelloftp;
import alice.tucson.api.*;
import alice.logictuple.*;

public class TupleTemplate {
  TupleCentreId tid;
  LogicTuple t=null;

  public TupleTemplate(TupleCentreId t) throws TucsonException{
    tid = new TupleCentreId(t.getName());
  }

  private boolean strInStr(String s1, String s2){
    int n = s2.length();
    do {
      if(s1.equals(s2)) return true;
      s1=s1.substring(1,s1.length());
    } while(s1.length()>=n);
    return false;
  }

  public LogicTuple createTupleTemplate(){
    if (strInStr(tid.getName(), "requests")){
      t = new LogicTuple("request", new Var(), new Var(), new Var(),
new Var());
    }
    if (strInStr(tid.getName(), "downloads")){
      t = new LogicTuple("download", new Var(), new Var(), new Var(),
new Var());
    }
    if (strInStr(tid.getName(), "errors")){
      t = new LogicTuple("error", new Var(), new Var(), new Var(), new
Var());
    }
    return t;
  }
}
```