UNIVERSITÀ DEGLI STUDI DI CATANIA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

FABRIZIO FAZZINO

VIRTUAL LAB:
UN AMBIENTE SICURO PER L'APPRENDIMENTO, IL MONITORAGGIO E LA GESTIONE REMOTA DI SISTEMI PER CONTROLLO DI PROCESSO ATTRAVERSO INTERNET.

Tesi di Laurea

Relatore:
Chiar.mo Prof. O. Mirabella
Correlatore:
Chiar.ma Prof.ssa A. Di Stefano

ANNO ACCADEMICO 1995-1996

1 - Introduzione al Laboratorio Virtuale

1.1 Prefazione

Grazie al rapido sviluppo dell'informatica e delle telecomunicazioni stanno oggi rendendosi disponibili varie tecnologie per rendere più semplice la vita di ogni giorno a vaste categorie di persone.

L'apprendimento a distanza è una realtà emergente grazie alla quale studenti, docenti e ricercatori possono essere situati presso differenti locazioni geografiche ed interagire ugualmente tra loro per partecipare alle lezioni e svolgere le esercitazioni.

Il controllo a distanza può permettere a personale specializzato di intervenire remotamente su di un impianto industriale per monitorarlo o per modificare il suo ciclo produttivo. Il tele-lavoro, più in generale, indica una attività lavorativa che può essere svolta dal lavoratore accedendo ai dati della propria società direttamente dalla propria abitazione.

Queste ed altre caratteristiche possono essere rese accessibili, con un progetto unitario, all'interno di un cosiddetto *Laboratorio Virtuale*, che sarà formalmente definito nel seguito. In questa tesi saranno analizzati i tanti aspetti del Laboratorio Virtuale, dalle varie scelte di progetto sino alle più complesse problematiche riguardanti la sicurezza; in ultimo, sarà presentata una completa implementazione di un Laboratorio Virtuale perfettamente funzionante ed accessibile tramite Internet da qualunque parte del mondo mantenendo dei requisiti di sicurezza molto stringenti.

L’aspetto più importante della presente trattazione è la generalità con cui verranno trattati i vari aspetti del Laboratorio Virtuale, cercando sempre di non ricondursi ad una applicazione specifica ma cercando di rendere il sistema progettato quanto più *general-purpose* possibile. Per mantenersi ad un livello di astrazione abbastanza elevato, tuttavia, non si adotterà la soluzione più semplice, ovvero quella di ignorare i vari problemi che potrebbero sorgere realizzando alcune particolari soluzioni, ma verranno analizzate esaustivamente ed in dettaglio tutte le varie possibilità implementative, valutando criticamente ad ogni passo i pro e i contro di ogni scelta.

1.2 Definizione

Un **Laboratorio Virtuale** (VL, *virtual lab*) è una delle nuove possibilità offerte dalle attuali tecnologie come ausilio per la didattica, la ricerca ed il tele-lavoro. Esso è un Sistema Distribuito che rende possibile ad un utente di accedere mediante Internet o Intranet ad un vero e proprio laboratorio remoto, potendo interagire a distanza con esso compiendo le normali operazioni che normalmente vengono eseguite in locale sui vari tipi di impianti.

Il concetto di Laboratorio Virtuale, applicato al controllo di sistemi industriali, potrebbe anche permettere di realizzare delle forme di monitoraggio e controllo remoto di impianti di qualunque tipo, facendo sì che i tecnici che hanno il compito di stabilire l’andamento del ciclo produttivo, oppure di diagnosticare e talvolta di risolvere eventuali malfunzionamenti dell’impianto, non debbano necessariamente essere presenti fisicamente nel luogo in cui sorge l’impianto da pilotare, ma possano idealmente accedervi da un qualunque punto del mondo mediante Internet.

Tale accesso agli impianti potrebbe essere accompagnato da altri servizi, e ad esempio sarebbe utile poter accedere dalla rete a dei tutorial che includano la documentazione relativa al funzionamento del particolare impianto in questione, in modo da fungere per il personale tecnico come riferimento per le manovre consentite.

Più in generale, però, il Laboratorio Virtuale oltre a porsi da tramite tra l'utente ed un impianto reale può incorporare altre funzionalità. Ad esempio se l'obiettivo dell'utente è quello di effettuare una campagna di misure su di un impianto sottoposto a determinati ingressi, al solo scopo di ottenere dei dati circa il comportamento di tale impianto per scopi didattici o di ricerca, i risultati delle esperienze potrebbero essergli forniti oltre che da un impianto vero anche da un opportuno simulatore. Anche per questo secondo tipo di utilizzo (didattico) del laboratorio è utile la presenza di tutorial consultabili on-line, in modo da istruire gli utenti circa le possibilità offerte dal simulatore.

Sia per gli impianti reali che per i simulatori è utile inoltre che gli accessi possibili per gli utenti non siano solo 'guidati', ovvero solo tramite interazioni con delle interfacce precostituite, ma si deve fare in modo che ogni singolo utente possa realizzare delle applicazioni proprie che utilizzino le risorse del laboratorio remoto.

1.3 Finalità

In definitiva il Laboratorio Virtuale deve offrire agli utenti l'accesso a tre tipi differenti di *risorse*: deve fungere da interfaccia verso **impianti reali**, sui quali possano essere eseguite tutte le normali operazioni di lettura delle variabili di stato, imposizione delle specifiche desiderate e opportuna configurazione dei riferimenti (comandi a distanza); deve offrire accesso a **simulatori** mediante passaggio dei parametri di ingresso, esecuzione remota e lettura

dei risultati ed inoltre anche l'accesso selettivo sulle singole porzioni di testo dei **tutorial** per la consultazione (lettura) e gli aggiornamenti (scrittura).

Le precedenti risorse devono poter essere utilizzate per le seguenti finalità:

1. **Remote Tutoring** (o *distance learning*, *apprendimento a distanza*): uno degli impieghi principali deve poter essere l'apprendimento a distanza, dunque si deve prevedere uno scopo didattico rivolto prevalentemente agli studenti che devono poter accedere a tutorial e simulatori dalle proprie abitazioni o dalle rispettive sedi di studio;
2. **Remote Control** (*controllo a distanza*): il personale tecnico addetto ad un impianto industriale, ma anche altre persone autorizzate, devono poter intervenire in tempo reale sull'impianto da una postazione remota per modificare le specifiche di produzione ed il tipo di controllo adoperato, nonché eventualmente per intervenire in caso di guasto; in questo caso è chiaro che le specifiche temporali di risposta del laboratorio devono essere molto stringenti;
3. **Remote Monitoring** (*monitoraggio a distanza*): le stesse persone autorizzate devono poter monitorare a distanza l'impianto in modo da valutarne off-line il comportamento nelle varie situazioni, ad esempio in regime di produzione normale od in presenza di guasti o malfunzionamenti;
4. **Remote Performance Evaluation** (*valutazione a distanza delle prestazioni*): col termine di valutazione a distanza si può intendere sia la valutazione on-line delle prestazioni di un impianto reale, da parte di tecnici o degli stessi studenti, sia la valutazione compiuta su di un analogo programma che simuli la presenza di un impianto dello stesso genere, riproducendone il comportamento nella maniera più fedele possibile.

1.4 Interfacce del laboratorio

Il laboratorio deve fornire due tipi di interfacce verso le risorse, rivolte rispettivamente all'utente ed al programmatore:

1. una **GUI** (*graphical user interface*, interfaccia grafica per l'utente), visuale ed intuitiva, in modo che anche per l'utente più inesperto sia immediato interagire in maniera 'guidata' con i vari oggetti che vengono proposti, semplicemente configurando opportunamente i parametri di ingresso ed osservando le corrispondenti uscite nella forma in cui vengono proposte;
2. inoltre dovrebbe anche essere presente una **API** (*application programming interface*, interfaccia per la programmazione delle applicazioni) che permetta all'utente di scrivere delle proprie applicazioni locali che interagiscano direttamente con le risorse del laboratorio, in modo che sia possibile assemblare tra loro varie risorse interfacciandosi con il VL a basso livello.

Nel realizzare questi servizi si desidera che tali interfacce soddisfino i seguenti requisiti:

1. **Upgrading Indipendence** (*indipendenza dagli aggiornamenti*): ogni eventuale modifica delle interfacce che il Laboratorio Virtuale offre all'esterno non deve comportare onerose operazioni di aggiornamento del software da parte di ogni utente.
2. **Platform Indipendence** (*indipendenza dalla piattaforma*): entrambe tali interfacce devono essere indipendenti dalla piattaforma hardware/software dell'utente remoto;

Riguardo al primo requisito, ovvero la garanzia di trasparenza rispetto a tutte le eventuali modifiche che si vogliano apportare al laboratorio, se si procedesse col normale ciclo acquisto-installazione-configurazione che si segue per poter utilizzare sul proprio computer i vari pacchetti software, tale approccio comporterebbe ad ogni modifica del software (nel

nostro caso le interfacce del Laboratorio Virtuale) una ridistribuzione e susseguente installazione della versione aggiornata da parte di tutti gli utenti; tale approccio non è chiaramente praticabile nel caso in cui si mediti di aggiornare e modificare continuamente i servizi offerti dal Laboratorio Virtuale agli utenti.

Allora per soddisfare il primo requisito si potrebbe pensare di adoperare un classico meccanismo di *telnet* remoto, con cui l'utente si collega tramite Internet con la stazione che gestisce il laboratorio ed esegue i comandi a distanza; però tale soluzione non è praticabile in quanto telnet offre spesso (ovvero su reti geografiche, come Internet) solo un accesso testuale remoto, e questo non consente di interagire mediante una GUI.

Non è infatti possibile prevedere neanche con altri meccanismi un tipo di accesso che preveda che l'interfaccia grafica visualizzata dall'utente sia eseguita da una macchina remota, in quanto le reti geografiche in generale, ed Internet in particolare, non sono in grado di garantire una banda sufficiente al trasporto di un flusso continuo di informazioni in formato grafico.

Fermo restando allora che le interfacce grafiche, per la loro stessa natura, devono necessariamente essere gestite dallo stesso computer su cui devono essere visualizzate, si potrà osservare che il metodo principale per implementare un programma che presenti all'utente una potente GUI consiste nell'utilizzare un linguaggio ad alto livello, magari orientato agli oggetti; però tali linguaggi (come il C++) non permettono di soddisfare nessuna delle due proprietà richieste, in quanto le librerie grafiche utilizzate sono fortemente dipendenti dalla piattaforma ed il software andrebbe reinstallato ad ogni modifica.

In questa tesi si è pensato allora di realizzare tutte le finalità prima esposte utilizzando il nuovo linguaggio Java sviluppato dalla Sun Microsystems [WHIT]. Java è un linguaggio object-oriented, fortemente ispirato al C++, che però presenta in più entrambi i requisiti richiesti per implementare il Laboratorio Virtuale:

1. con Java è possibile realizzare dei piccoli programmi (*applet*) che possono essere richiamati da un utente remoto mediante un comune *browser* World Wide Web, dal quale vengono interpretati ed eseguiti in locale; così l'utente ha bisogno del solo browser, e ad ogni collegamento gli verrà automaticamente presentata l'interfaccia più aggiornata per interagire con il laboratorio;
2. il linguaggio Java dispone di librerie grafiche indipendenti dalla piattaforma, e dunque una applet o applicazione Java può essere eseguita indifferentemente su computer con sistemi operativi Win32 (NT o 95), OS/2, MacOS e su varie piattaforme della famiglia Unix (prima tra tutte Solaris della stessa Sun).

1.5 Realizzazione proposta

Grazie alla presenza di Web browser Java-enabled è possibile fornire a tutti gli utenti l'interfaccia GUI con il Laboratorio Virtuale senza fornire loro fisicamente il programma, ma scaricando sulla macchina locale la versione più aggiornata solo al momento della connessione tramite il proprio browser.

Così il generico utente connesso ad Internet per instaurare il collegamento con il laboratorio remoto necessita solamente di un Web browser Java-enabled (come Netscape Navigator) e dell'indirizzo HTTP (HyperText Transfer Protocol) del laboratorio stesso.

Una volta effettuata la connessione, sulla macchina locale dell'utente viene trasferita l'interfaccia grafica che permette all'utente di interagire con le risorse messe a disposizione dal laboratorio.

Invece per quanto riguarda la API il meccanismo di interazione è più complesso, in quanto bisogna fornire agli utenti gli strumenti, completi di tutta la documentazione, per fare in

modo le applicazioni da essi sviluppate possano colloquiare con le risorse del VL anche senza utilizzare il browser e le interfacce precostituite della GUI.

Tali possibilità offerte dal linguaggio Java permettono di realizzare delle implementazioni che soddisfino i due requisiti di indipendenza prima esposti, permettendo soprattutto all'utente un accesso rapido e semplice; deve però essere chiaro che tutta l'analisi del Laboratorio Virtuale che verrà fatta nei prossimi capitoli non presuppone l'utilizzo di un particolare linguaggio di programmazione, e che solo la particolare implementazione proposta alla fine fa utilizzo delle caratteristiche offerte dal linguaggio Java.

1.6 Sezioni

La tesi si compone essenzialmente di tre sezioni:

1. introduzione dei **problemi generali** legati ad Internet e alla sicurezza in genere;
2. analisi dei problemi connessi alla **progetto di un Laboratorio Virtuale** e possibili risoluzioni;
3. **implementazione** del Laboratorio Virtuale e documentazione delle interfacce offerte all'utente.

La **prima sezione** di questa tesi offre una panoramica dei concetti generali nel campo delle reti di computer e della sicurezza dei sistemi distribuiti che costituiscono la base per la ricerca affrontata. In particolare nel seguente capitolo 2 verranno introdotti i concetti generali del mondo Internet, spiegando i termini già adoperati come 'browser', 'Web' e 'HTTP', nonché il modello Client/Server. Nel capitolo 3 si introdurrà il linguaggio Java, analizzando il concetto di 'applet' e le caratteristiche utili alla realizzazione del Laboratorio Virtuale. Nel

capitolo 4 si vedranno i principali requisiti di sicurezza delle informazioni, le varie tipologie di attacchi e le possibili soluzioni attualmente di uso comune. Nel capitolo 5 si vedrà come sia possibile creare un modello concettuale di alto livello, detto 'Modello di Sicurezza', che consenta di implementare varie politiche di controllo sugli accessi da parte degli utenti. Infine nel capitolo 6 viene definito il concetto di Sistema Distribuito e saranno presentate le varie possibilità di realizzazione attualmente esistenti..

La **seconda sezione** inizia dal capitolo 7 che applica il concetto di Sistema Distribuito ad un caso concreto per definire l'architettura del Laboratorio Virtuale; tale caratterizzazione prosegue nel capitolo 8, in cui vengono analizzate le varie possibilità di organizzazione del server, e nel capitolo 9, in cui vengono viste nel dettaglio le modalità con cui il laboratorio effettua l'accesso alle varie risorse. Successivamente nel capitolo 10 vengono introdotti i problemi generali di sicurezza che vengono a presentarsi durante il progetto di un VL. Tali problemi vengono particolareggiati nel capitolo 11, con la presentazione di un protocollo di autenticazione per identificare gli utenti remoti che si collegano col laboratorio tramite Internet, e nel capitolo 12, in cui si cerca di realizzare un modello di sicurezza che sia adatto ad effettuare varie politiche di controllo sugli accessi al VL da parte degli utenti. Come conclusione della parte relativa alla sicurezza vengono viste, nel capitolo 13, le varie modalità di attacco alla sicurezza del laboratorio e dell'utente.

Infine nella **terza sezione**, che inizia dal capitolo 14, vengono presentate nel dettaglio le varie scelte implementative che conducono ad utilizzare il linguaggio Java per la realizzazione del Laboratorio Virtuale, con varie implicazioni sull'architettura da adottare. Nel capitolo 15 ci si sofferma sulla API disponibile per il programmatore e sul formato delle comunicazioni tra il Client ed il Server, nel capitolo 16 si analizzano la caratteristiche di sicurezza e le prestazioni offerte da questa implementazione di Laboratorio Virtuale e nel capitolo 17 si traggono le conclusioni sulla effettiva utilizzazione ed innovazione di quanto realizzato. Infine, nei capitoli

dal 18 al 22, vengono riportati i sorgenti intergrali del Laboratorio Virtuale totalmente implementato e funzionante.

2 - Nozioni di Internetworking

2.1 Internet

Internet è una rete di computer di portata mondiale che, partendo dalle quattro università statunitensi che nel 1969 parteciparono alla versione originale di ARPAnet, si compone oggi di migliaia di singole reti, ciascuna che raccoglie a sua volta un numero più o meno grande di *host*.

Un **host** è un singolo computer collegato alla rete ed identificabile attraverso un proprio indirizzo, e tipicamente denota una macchina di una certa potenza di calcolo e di memorizzazione.

Invece un insieme di reti interconnesse tra loro ma non collegate ad Internet, cioé considerate solo entro l'ambito privato dell'azienda in cui sono state costruite oppure di un gruppo di aziende inaccessibili al pubblico comune, assume il nome di **Intranet**.

Internet ed Intranet utilizzano come suite di protocolli di comunicazione lo **stack TCP/IP** (*Transmission Control Protocol / Internet Protocol*), che per sua natura è tale da consentire l'interconnessione delle reti più eterogenee, dalle LAN convenzionali (come Ethernet) alle reti geografiche che impiegano linee telefoniche. Tutti i computer al mondo dispongono del loro stack TCP/IP, a partire dai personal computer per arrivare ai mini computer e ai grandi mainframe dei centri di calcolo.

Ogni applicazione che utilizza la rete trasmette e riceve i dati che manipola attraverso delle **porte** identificate da un numero; tra due porte di due qualunque computer di Internet

possono essere effettuate delle connessioni dette **socket**. Molti linguaggi di programmazione includono delle librerie tramite cui dei processi su due diverse macchine di una rete (anche Internet) possono essere fatti colloquiare; in questi casi solitamente una delle due parti si pone in attesa che l'altra effettui una richiesta, ed in questo caso si parla di *modello Client/Server*.

2.2 Il modello Client/Server

Il modello standard per le applicazioni di rete è il **modello Client/Server** [STEV]. Un *server* è un processo in attesa di essere contattato da un processo *client* per svolgere per esso qualche servizio.

Il processo server viene avviato su un determinato host, e rimane in attesa delle richieste di servizio che possono giungere da qualunque altro nodo della rete; i processi client vengono solitamente inizializzati da utenti interattivi che richiedono qualche servizio sul proprio sistema.

Le richieste di servizio dei client e le relative risposte dei server possono idealmente viaggiare su qualunque rete di computer, e dunque tale modello si addice anche ad Internet.

Una distinzione importante che è possibile fare riguarda il tempo richiesto dal processo server per onorare le richieste dei client:

- se la richiesta del client può essere servita dal server in un breve intervallo di tempo, il processo server gestisce da sé la richiesta e viene detto **server iterativo**;
- se invece l'intervallo di tempo per servire una richiesta può essere indefinitamente lungo, il server deve chiamare un altro processo per gestire ogni richiesta dei client, cosicché il processo server originale può tornare a dormire in attesa della successiva richiesta di un

client. Naturalmente in tal caso si richiede che il sistema operativo consenta l'esecuzione simultanea di più processi, e si parla di **server concorrenti**.

2.3 Il World Wide Web

Il **World Wide Web** (detto WWW o semplicemente Web) è uno dei principali servizi offerti da Internet, e permette di stabilire collegamenti tra documenti ipertestuali memorizzati su computer diversi, detti **Web server**. In questo modo un utente, durante la consultazione, può saltare da un documento all'altro, senza curarsi di dove si trovino fisicamente i relativi server.

Il programma di lettura delle pagine Web è detto **Web browser**, o più semplicemente browser, ed è un programma che funziona sulla macchina dell'utente (che dunque rappresenta il lato client) e permette di navigare sulla rete Web esplorandone i contenuti. Attualmente dunque ad un utente che voglia accedere ad una qualunque pagina Web presente su qualunque host del mondo è sufficiente munirsi di un collegamento Internet e di un browser; fortunatamente i browser sono disponibili per qualunque sistema operativo, occupano poco spazio su disco e ne esistono diverse versioni *freeware*.

Il protocollo usato dal Web per il trasferimento dei documenti ipertestuali è l'**HTTP** (*HyperText Transfer Protocol*). I documenti Web, detti pagine, possono contenere sia testo che grafica, il tutto coordinato per mezzo di uno speciale linguaggio di codifica dei vari oggetti chiamato **HTML** (*HyperText Markup Language*). Questo linguaggio per la codifica degli ipertesti è costituito da un gruppo di codici che indicano al programma come visualizzare la pagina e permettono di stabilire collegamenti tra una parola e un'altra pagina Web, magari memorizzata su un server remoto. Il testo da codificare deve essere in formato ASCII e la

marchiatura dei brani interessati dal collegamento ipertestuale avviene tramite l'aggiunta di opportuni *tag* (etichette).

Lo **URL** (*Uniform Resource Locator*) è un metodo standard per indicare l'indirizzo logico di una specifica risorsa di Internet, ad esempio una pagina Web (ovvero da trasferire col protocollo HTTP) o un server di trasferimento file (FTP, *file transfer protocol*). Il termine URL è anche sinonimo dell'indirizzo stesso, che è composto da una stringa di caratteri senza spazi che identifica in modo univoco qualunque risorsa esistente su Internet. Il formato dell'indirizzo è il seguente:

protocollo://host:porta/percorso

dove `protocollo` è ad esempio uno tra FTP, HTTP, Gopher o Telnet; `host` è il nome simbolico del particolare computer su cui la risorsa risiede; `porta` è un numero che indica il tipo di servizio richiesto all'host, e per default vale 80 e richiede servizi al Web server; infine `percorso` indica la posizione del file in relazione alle directory esistenti sull'host in questione.

Un esempio di URL potrebbe essere *http://www.unict.it/vlab/* .

3 - Il linguaggio Java

3.1 La macchina virtuale di Java (JVM)

Il linguaggio Java è stato sviluppato dall'inizio degli anni '90 dalla Sun Microsystems. Esso è un linguaggio di programmazione totalmente orientato agli oggetti che presenta alcune caratteristiche che lo rendono particolarmente innovativo. In realtà Java, molto simile al C++, è stato progettato ispirandosi a vari linguaggi, tra i quali SmallTalk, Ada e Objective C, prendendo da ognuno gli aspetti più interessanti.

La prima caratteristica particolare di Java è che si tratta di un **linguaggio interpretato**, cioé il codice sorgente non viene compilato direttamente in formato binario (ovvero direttamente eseguibile dal microprocessore e dal sistema operativo per cui è stato destinato); bensì il sorgente viene trattato con una sorta di precompilatore che dà luogo al codice in formato *byte-code*, che è una sorta di codice eseguibile al di sopra di una certa 'macchina virtuale'. Solo al momento dell'esecuzione viene eseguito l'interprete del linguaggio Java, che provvede run-time ad interpretare ed eseguire tale codice al di sopra della **Java Virtual Machine** (*JVM*), ed in tal modo il byte-code è identico per tutti i sistemi sui quali sia stato sviluppata una macchina virtuale.

La macchina virtuale è cioé un eseguibile differente per ogni piattaforma, ma presenta al di sopra di esso una interfaccia identica che permette di eseguire qualunque applicazione o applet gia in formato byte-code. La JVM è una sorta di computer generico basato sulle

specifiche dell'interfaccia POSIX, per cui l'implementazione della macchina virtuale su nuove architetture a 32 bit è abbastanza semplice a condizione che siano supportati i processi multipli.

Dunque Java è un linguaggio interpretato e dinamico, in quanto l'interprete Java esegue le istruzioni nel proprio codice, ed il linking avviene solo al momento dell'esecuzione. Nell'ambiente di sviluppo non c'è una fase di linking separata, ma viene sostituita dal processo di caricamento di nuove classi da parte del *ClassLoader*. Le classi vengono linkate solo quando è necessario e possono essere scaricate dalla rete.

La caratteristica dell'interpretazione, con l'introduzione della macchina virtuale, è normalmente un aspetto negativo per un linguaggio di programmazione, in quanto l'esecuzione di un programma risulterà certamente più lenta di quella di un codice in binario. In realtà tale proprietà è stata introdotta a causa del grande sviluppo delle telematica e ha portato all'introduzione di un nuovo concetto di programmazione.

Infatti con Java è possibile scrivere dei programmi, detti **applet**, che possono essere inviati sulla rete dai Web server ed eseguiti, grazie all'interpretazione, sui Web browser delle macchine che li ricevono. In tale modo i programmi possono circolare sulla rete come se siano dei normali documenti Web, ed essere sempre eseguiti in locale con grandi vantaggi sia per quanto riguarda la reperibilità (non c'è bisogno di cercare ed acquistare un programma, basta scaricarlo automaticamente ed eseguirlo ogni volta che serve), sia per quanto riguarda la velocità di esecuzione di applicazioni in modalità grafica rispetto ai metodi tradizionali come *telnet*.

Inoltre, mentre il fatto che il linguaggio sia interpretato consente la spedizione sulla rete e la successiva esecuzione remota, entro breve dovrebbe perdersi anche lo svantaggio della tradizionale lentezza di esecuzione dei programmi scritti mediante linguaggi interpretati piuttosto che compilati: infatti Sun Microsystems sta pianificando la realizzazione di una serie

di microprocessori che supportino le specifiche della JVM e che siano particolarmente efficaci nell'esecuzione del codice Java.

3.2 Robustezza

Una caratteristica particolare di Java è che si tratta di un linguaggio particolarmente sicuro. Infatti, pur essendo ispirato esplicitamente al C++, rimuove da esso tutte le caratteristiche di basso livello (come i puntatori) che possono rendere possibile un accesso diretto alla memoria del proprio computer. Un programma Java è, per così dire, 'guidato', per cui anche eseguendo una applet scaricata da Internet l'utente può stare certo che tale programma non potrà eseguire nessuna operazione fraudolenta.

Così, sebbene Java sia molto simile al C++ per il programmatore, esso rimuove dal C++ molte delle caratteristiche che potevano generare confusione e condurre ad errori (aritmetica dei puntatori, strutture, `typedef`, `#define` e necessità di liberare la memoria).

Il linguaggio Java è inoltre reso più robusto dal fatto che effettua molti controlli sia in fase di compilazione che in fase di esecuzione; ad esempio fornisce la **gestione delle eccezioni**, in maniera molto simile al linguaggio Ada, e la valutazione dei limiti degli array evitando run-time accessi con indici fuori dai limiti (cosa invece molto frequente in C e causa di malfunzionamenti molto difficili da diagnosticare).

Inoltre il codice Java è sicuro perché in fase di esecuzione viene testato da un **Byte-Code Verifier** che controlla che il byte-code non causi overflow dello stack, che il tipo dei parametri passati sia corretto e che non avvengano conversioni illegali tra tipi di dati.

3.3 Portabilità

Un’altra caratteristica di Java è il fatto di essere un linguaggio dotato sin dalla nascita di una serie di librerie che comprendono tutte le classi e i metodi fondamentali, compresi quelli per la gestione di una interfaccia a finestre (**AWT**, *abstract windowing toolkit*) anch’essa multipiattaforma.

Dunque le applicazioni Java vengono ad essere neutrali rispetto all’architettura, possono cioé essere eseguite su qualunque architettura per la quale esista la macchina virtuale Java. Inoltre, per aumentare la portabilità, oltre alla neutralità rispetto all’architettura viene fatto in modo che anche la rappresentazione interna dei dati (ad esempio numeri interi od in virgola mobile) sia identica su tutte le piattaforme.

3.4 Esecuzione distribuita e multithreading

Come evidenziato ripetutamente in [WHIT] e [PROG], il linguaggio Java presenta infine due ulteriori vantaggi rispetto agli altri linguaggi; innanzitutto esso è **distribuito**, ovvero incorpora la capacità di utilizzare lo stack di protocolli TCP/IP, nonché altri protocolli, come ad esempio HTTP ed FTP, per reperire informazioni lungo la rete.

Inoltre è **multithreaded**, cioé è possibile suddividere le varie attività di un programma in più thread indipendenti, in modo che ognuno possa essere terminato indipendentemente dagli altri.

3.5 Strumenti di sviluppo

La Sun Microsystems provvede personalmente a sviluppare e distribuire gratuitamente il *Java Developers Kit* (**JDK**) sia nella versione Solaris che in quella Win32, ed in più altri produttori di software, come Apple e IBM, hanno prontamente acquistato i diritti per portare Java anche sulle proprie piattaforme MacOS, OS/2 e AIX.

Nella sua versione base, distribuita gratuitamente, il JDK, ovvero il kit di Java per lo sviluppatore, comprende il compilatore in formato byte code (`'javac'`), l'interprete run-time (`'java'`), un debugger a riga di comando (`'jdb'`), un programma di utilità per interfacciare un programma Java con delle routine in codice in linguaggio C nativo (`'javah'`), un programma per eseguire le applet (`'appletviewer'`), nonché la libreria completa delle classi di Java.

Anche questo ambiente di sviluppo di Java è facilmente portabile verso nuove architetture o sistemi operativi, in quanto il compilatore è scritto in Java ed il sistema di run-time in ANSI C.

3.6 Applicazioni e Applet

Come già accennato, con Java è possibile scrivere delle applet, ovvero dei programmi che possono essere collegati con delle pagine Web ed essere eseguiti dal proprio browser; questo però non preclude la possibilità di scrivere delle applicazioni separate.

I programmi scritti mediante il linguaggio Java si dividono allora in due categorie [LANG, PROG] :

- **applicazioni:** sono dei programmi *stand-alone* che vengono tipicamente eseguiti su una macchina nella maniera classica; rispetto a tutti gli altri linguaggi ad alto livello l'unica

differenza è che il linguaggio Java non è compilabile in binario ma richiede la presenza di un interprete.

- **applet:** sono dei programmi che possono essere collegati ad una pagina HTML per essere inviati sulla rete ed eseguiti (interpretati) dal browser della macchina richiedente.

Una applicazione viene scritta normalmente, come si fa abitualmente per un programma in C++, includendo ogni classe in un file sorgente differente, ed avendo cura che la classe da cui dovrà partire l'esecuzione del programma includa un metodo `main( )` . Il sorgente `nome.java` compilato dà origine al byte code `nome.class`, che andrà eseguito col comando `java nome`.

Invece una applet va scritta estendendo la classe *Applet* disponibile nella libreria di Java, e ridefinendo i suoi metodi di base, ovvero `init( ), start( ), stop( ), destroy( ), paint( )` e `update( )`. Tale applet va poi collegata ad una pagina HTML (mediante il tag `<APPLET>`) che va poi richiamata da `appletviewer` o dal proprio browser.

Si è sottolineata la differenza tra i due tipi di programmi Java in quanto saranno entrambi necessari per la realizzazione del Laboratorio Virtuale: infatti, come sarà analizzato nel seguito, quando verrà descritto il modello Client/Server per la comunicazione tra l'utente ed il laboratorio, si vedrà come il server non sia altro che una applicazione che gira sulla macchina su cui è implementato il laboratorio, mentre il client utilizzato dall'utente è una applet che viene fatta eseguire sulla propria macchina grazie ad un opportuno browser.

4 - Sicurezza delle informazioni

4.1 Security

I file system dei computer contengono spesso informazioni che vanno protette da utenti non autorizzati. Con la terminologia corrente, si parla di **security** quando ci si riferisce ai problemi generali legati alla prevenzione dell'accesso non autorizzato a tali informazioni; più in particolare si parla di **politiche di sicurezza** (*policies*) quando ci si riferisce alla definizione delle regole astratte in base alle quali l'accesso deve essere consentito ad alcuni utenti e negato ad altri, quando invece si parla di **meccanismi di protezione** (*protection mechanisms*) in riferimento agli specifici meccanismi utilizzati in concreto per salvaguardare tali informazioni [TANE].

I principali *requisiti di sicurezza* richiesti ad un sistema di elaborazione sono solitamente i seguenti [STAL] :

- **Confidentiality** (*confidenzialità*): le informazioni contenute nel sistema devono poter essere lette solo dalle parti autorizzate.
- **Authentication** (*autenticazione*): l'origine di ogni messaggio deve poter essere identificata con certezza.
- **Integrity** (*integrità*): le informazioni devono poter essere modificate solo dalle parti autorizzate.
- **Nonrepudiation**: né il trasmittente di un messaggio né il ricevente possono negare che sia avvenuta la trasmissione.

- **Availability** (*disponibilità*): le risorse devono essere disponibili alle parti autorizzate quando richieste.

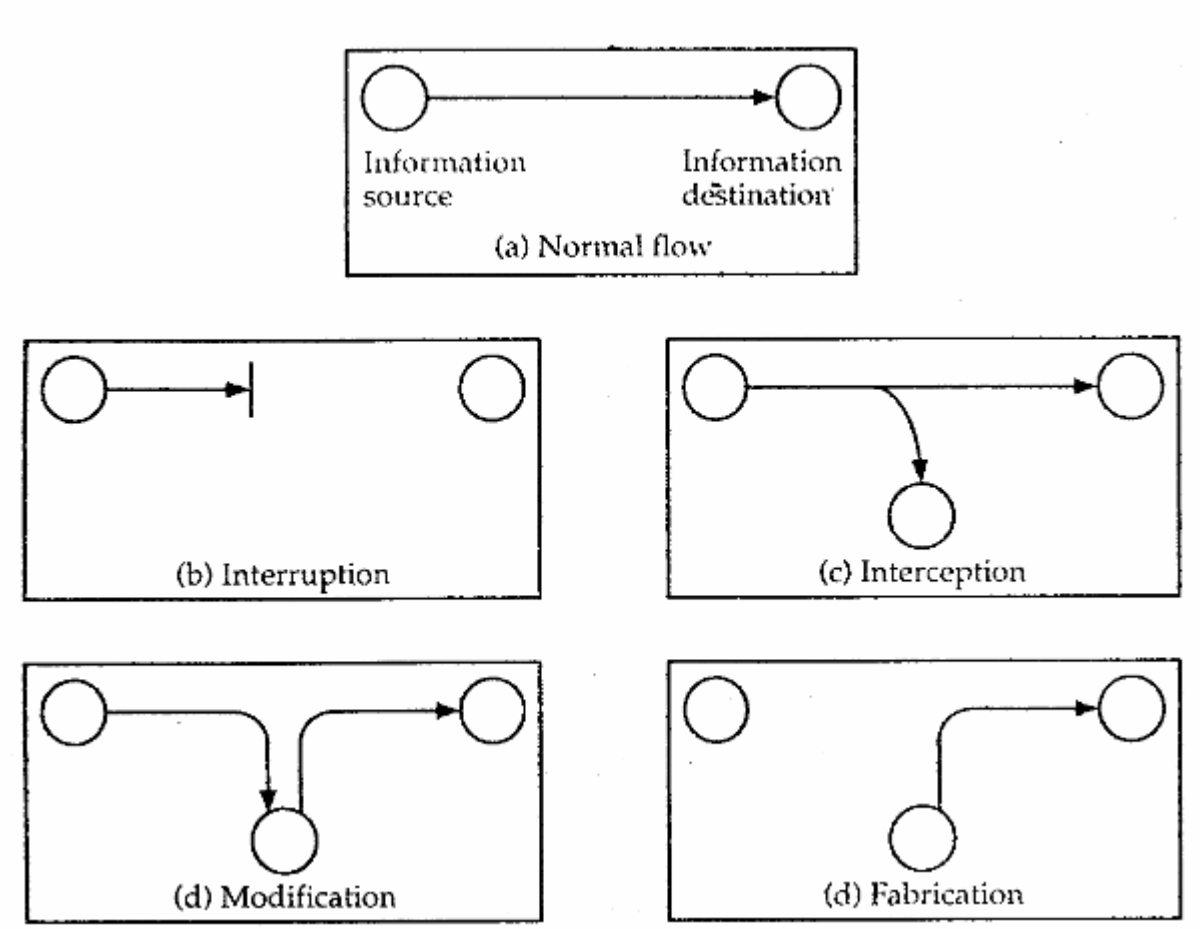
A questi requisiti va aggiunto spesso anche l'**Access Control** (*controllo degli accessi*) sotto il cui nome vanno tutti i meccanismi di protezione adoperati per tenere nota degli utenti che si sono collegati al sistema.

4.2 Attacchi alla sicurezza

In ogni sistema di elaborazione si hanno flussi di informazioni; in particolare nelle reti di computer si hanno sempre dei messaggi che viaggiano da un host ad un altro lungo delle linee trasmissive.

A seconda delle modalità con cui agiscono, possono verificarsi 4 differenti categorie (per ora generiche) di attacchi alla sicurezza [STAL]:

1. **Interruption** (*interruzione*): una risorsa del sistema viene resa non disponibile (ad esempio un messaggio inviato sulla rete viene distrutto); è un attacco contro la availability.
2. **Interception** (*intercettazione*): una parte non autorizzata guadagna accesso ad una risorsa (ad esempio leggendo un messaggio in transito); è un attacco contro la confidentiality.
3. **Modification** (*modifica*): una parte non autorizzata non solo guadagna accesso ad una risorsa ma la manomette (ad esempio modificando parte di un messaggio inviato sulla rete); è un attacco contro l'integrity.
4. **Fabrication**: una parte non autorizzata inserisce delle informazioni contraffatte nel sistema (ad esempio inviando sulla rete dei nuovi messaggi); è un attacco contro la authenticity.



Di questi attacchi solo l'interception è un **attacco passivo**, in quanto ci si limita ad ascoltare la trasmissione per carpire delle informazioni; allora un attacco di questo genere è molto difficile da individuare, in quanto non produce nessun effetto sul sistema attaccato, ma per fortuna esistono metodi abbastanza semplici per prevenire che ciò accada (come ad esempio l'utilizzo della cifratura delle informazioni, che vedremo più avanti).

Invece gli altri tre tipi sono **attacchi attivi**, cioé comportano modifiche sul flusso di dati o la creazione di nuovi messaggi; al contrario degli attacchi passivi, gli attacchi attivi sono molto facili da individuare ma difficili da prevenire (anche se la stessa individuazione può fungere da deterrente).

A loro volta gli attacchi attivi possono essere suddivisi in 4 ulteriori categorie:

- **Masquerade**: una entità pretende di essere un'altra.

- **Replay** (*ritrasmissione*): messaggi catturati passivamente vengono in seguito ritrasmessi per produrre effetti non autorizzati.
- **Message Modification** (*modifica dei messaggi*): i messaggi trasmessi vengono alterati, riordinati o ritardati per produrre effetti non autorizzati.
- **Denial of Service** (*negazione del servizio*): viene proibito l'utilizzo del sistema da parte degli utenti autorizzati, ad esempio distruggendo tutti i messaggi.

4.3 Intrusi

Gli intrusi (*intruder*, detti anche *hacker* o *cracker*) sono la principale fonte di attacchi alla sicurezza di un sistema. Essi possono venire classificati in 3 categorie:

1. **Masquerader**: individuo non autorizzato ad usare il sistema che vi penetra sfruttando l'account di un utente legittimo.
2. **Misfeasor**: utente legittimo del sistema che però abusa dei propri diritti accedendo a risorse per le quali non è autorizzato.
3. **Clandestine User**: individuo che si impadronisce del controllo della supervisione del sistema per evitare la verifica degli accessi (*audit*).

Di solito il primo tipo di intruso è esterno al sistema (*outsider*), il secondo è interno (*insider*) ed il terzo può essere l'uno o l'altro.

4.4 Autenticazione degli utenti

Molti schemi di protezione si basano sull'assunzione che il sistema conosca l'identità di ogni utente. Come già visto tra i requisiti di sicurezza, si dice autenticazione la proprietà grazie alla quale è possibile assicurarsi dell'origine di ogni messaggio; più in particolare si dice **autenticazione dell'utente** (*user authentication*) il problema dell'identificazione degli utenti che si connettono, in modo da poter successivamente fornire ad essi dei permessi per compiere determinate operazioni; infatti normalmente l'autenticazione è finalizzata al conferimento all'utente remoto delle autorizzazioni (*authorization*) sulle prerogative di accesso sul sistema ospite.

L'autenticazione dell'utente consiste nel confrontare le informazioni fornite dalla persona con quelle memorizzate nel proprio archivio; nel caso che le informazioni coincidano, si può supporre che l'utente remoto sia quello che dice di essere.

La forma di autenticazione usata in maniera più diffusa è quella di richiedere all'utente di digitare una parola d'ordine (*password*); è evidente come tale metodo sia esposto al rischio di intercettazione da parte di un intruso, ed è per questo che i meccanismi di autenticazione sono sempre accompagnati da metodi crittografici.

4.5 Crittografia

Le informazioni in transito sulla linea fisica di collegamento tra due computer possono purtroppo essere soggette ad intercettazioni. Soprattutto in considerazione del fatto che sul percorso di trasmissione possono essere impiegati collegamenti via satellite, e dunque i dati possono essere letti da chiunque, è necessario un metodo di cifratura per rendere i dati inintellegibili a tutti tranne che al destinatario.

L’arte di cifrare i messaggi per renderli sicuri è detta **crittografia** (o più semplicemente *cifratura*). L’arte di rivelare il contenuto di codici segreti è definita *crittoanalisi*. L’arte di escogitare codici segreti e di rivelarli, cioé l’unione della crittografia e della crittoanalisi, è complessivamente nota come *crittologia*.

Il messaggio da cifrare, detto **testo in chiaro** (*plaintext*), viene convertito in una forma codificata equivalente, detta **testo cifrato** (*ciphertext*), attraverso un algoritmo di cifratura (*encryption*). Il testo cifrato viene quindi trasmesso al destinatario, che ricostruisce il messaggio originario grazie ad un altro algoritmo di decifrazione (*decryption*).

In genere gli algoritmi di cifratura e decifrazione sono parametrizzati da una *chiave*, ovvero una stringa di caratteri o di bit, ed è proprio grazie alla conoscenza di questa chiave segreta che il destinatario riesce a decifrare il messaggio. Così, se sulla linea di trasmissione è in ascolto un intruso che ricopia accuratamente il testo cifrato completo, egli non può decifrare il messaggio perché non conosce la chiave.

Una regola fondamentale della crittografia è che bisogna presupporre che il crittoanalista intruso conosca il metodo generale di cifratura impiegato, e tutta la segretezza dell’algoritmo viene fatta ricadere sulla chiave. Infatti, contrariamente all’algoritmo, le chiavi possono essere facilmente cambiate con la frequenza necessaria a non lasciare il tempo al crittoanalista di ricavarle.

La crittografia può essere in generale impiegata per raggiungere i seguenti obiettivi:

1. protezione dei dati contro la lettura da parte di persone non autorizzate (*confidentiality*);
2. proibizione a persone non autorizzate di inviare messaggi (infatti un messaggio non cifrato correttamente appena decifrato risulta privo di senso);
3. verifica dell’autenticità dell’identità del trasmettitore di ciascun messaggio (*autenticazione*);
4. possibilità di inviare elettronicamente documenti firmati (*firma elettronica*).

4.6 Algoritmi di cifratura dei dati

Le regole tramite cui due parti remote si accordano, si mettono in comunicazione tra loro e riescono a scambiarsi informazioni opportunamente codificate sono dette **protocollo**. In particolare nel caso della cifratura dei dati il protocollo sancisce le modalità con cui due entità remote possono scambiarsi informazioni riservate se

Un protocollo sicuro per lo scambio di tali informazioni tra le due entità, considerato che potrebbero esservi terze parti in ascolto sulla connessione fisica tra i due computer, necessita di due algoritmi di cifratura, uno simmetrico e l'altro asimmetrico (detto "a chiave pubblica").

Un **algoritmo simmetrico** è quello in cui un messaggio cifrato utilizzando un certo algoritmo matematico ed una certa chiave viene decifrato usando lo stesso procedimento e la stessa chiave, cioé la cifratura ed la decifrazione inversa coincidono.

Per questo tipo di algoritmi tutta la sicurezza risiede nell'unica chiave utilizzata per la comunicazione, allora la fase critica è quella in cui le due entità che vogliono comunicare si accordano sulla chiave da utilizzare, in quanto tale chiave non può essere inviata in chiaro sulla rete per non essere soggetta ad intercettazione da parte di intrusi in ascolto sulla connessione fisica tra le due macchine.

Tale algoritmo è quello che può essere utilizzato 'a regime' tra le due macchine, una volta che sia stata completata una fase preliminare di connessione e scambio di alcuni parametri.

I più diffusi algoritmi simmetrici sono il DES (*Data Encryption Standard*) e l'IDEA (*International Data Encryption Algorithm*), che cifrano blocchi di 64 bit utilizzando delle chiavi rispettivamente di 56 e 128 bit.

L'IDEA è probabilmente un algoritmo più sicuro del DES (la sola chiave di lunghezza più che doppia potrebbe bastare a confermare l'affermazione) [SCHN], ma quest'ultimo ha dalla sua il vantaggio di essere stato internazionalmente assunto come algoritmo standard per la cifratura dei dati, per cui per il Laboratorio Virtuale si preferisce adeguarsi alla comunità internazionale ed adottare il DES.

Un **algoritmo asimmetrico a chiave pubblica** è invece un algoritmo di crittografia di cui può essere resa nota la chiave da utilizzare per la cifratura, ma è impossibile (con la capacità di calcolo attualmente a disposizione) risalire da questa alla chiave da usare per la decifrazione (che impiega un algoritmo differente da quello della cifratura).

In questo tipo di algoritmi è dunque necessaria una coppia di chiavi (*chiave pubblica / chiave privata*) per effettuare la comunicazione tra due entità, e queste due chiavi devono essere gestite dall'entità che deve ricevere il messaggio. Infatti questa entità può rendere nota la chiave pubblica, che viene così utilizzata dall'entità trasmittente per cifrare il messaggio e spedirlo all'entità ricevente, e quest'ultima è l'unica in grado di decifrarlo in quanto è l'unica a possedere la chiave privata.

Nel caso in cui si desideri implementare una comunicazione bidirezionale mediante un algoritmo asimmetrico, entrambe le entità dovrebbero possedere la propria coppia di chiavi pubblica/privata, in modo da comunicare in chiaro quella pubblica ed utilizzare quella privata per la decifrazione. In realtà questa modalità è raramente utilizzata, in quanto si preferisce utilizzare un algoritmo asimmetrico solo per effettuare lo scambio di una chiave comune e proseguire la comunicazione con un algoritmo simmetrico.

Questo algoritmo è allora quello che nel laboratorio può essere utilizzato all'inizio della comunicazione per fare scambiare tra le due parti interessate la chiave da utilizzare per il successivo algoritmo simmetrico; è sufficiente una sola coppia di chiavi pubblica/privata gestite dal server del laboratorio in quanto sarà sufficiente che il server stabilisca una chiave simmetrica e la comunichi al client interessato ad instaurare una connessione.

Gli algoritmi asimmetrici si basano su proprietà matematiche che rendono possibile che il processo di decifrazione sia molto più complesso di quello di cifratura dei dati. La proprietà che in tali algoritmi viene sfruttata più di frequente è la difficoltà della fattorizzazione di numeri molto grandi, e tipicamente il processo di generare una coppia di chiavi pubblica/privata si riconduce alla generazione di numeri primi della lunghezza di svariate centinaia di bit (es. 512 bit) che soddisfino determinate caratteristiche. E' evidente come tali numeri siano mal gestibili in software (in quanto superano di gran lunga la lunghezza massima consentita dai *long int* nei linguaggi di alto livello), ed il solo approccio possibile è quello probabilistico (ovvero si cerca di non fattorizzare il numero, ma di generare tanti numeri della lunghezza desiderata finché qualche opportuno test indica che ci siano elevate possibilità che il numero in questione sia primo a uno).

Il più noto algoritmo di crittografia a chiave pubblica è RSA (detto così dai cognomi degli autori, Rivest, Shamir e Adleman), ma il più facile da implementare in software è l'algoritmo Knapsack che non richiede la generazione di numeri primi a uno di elevata lunghezza, ma si basa solo sulla conoscenza o meno di opportune sequenze di numeri (*superincreasing knapsacks*) che fungono da chiave privata e da cui possono essere ricavate altre sequenze da utilizzare come chiavi pubbliche. Inoltre RSA è gravato da diversi brevetti statunitensi che ne limitano l'utilizzo.

4.7 L'algoritmo DES

Il Data Encryption Standard è un algoritmo di crittografia simmetrico sviluppato dalla IBM ed adottato nel 1977 dal governo degli USA come standard ufficiale di cifratura delle informazioni.

Il testo in chiaro viene cifrato in blocchi di 64 bit, che producono 64 bit di testo cifrato. L'algoritmo, che è parametrizzato da una chiave di 56 bit, ha 19 stadi distinti. Esso è stato progettato per essere simmetrico, ovvero consente la decifrazione utilizzando la medesima chiave utilizzata per la cifratura, solo che i passi sono effettuati nell'ordine inverso.

Il primo stadio è una trasposizione indipendente dalla chiave del testo in chiaro; l'ultimo stadio svolge esattamente la trasposizione inversa. Il penultimo stadio scambia i 32 bit più a sinistra con i 32 bit più a destra. I restanti 16 stadi sono funzionalmente identici ma sono parametrizzati da differenti funzioni della chiave.

Ognuno di questi 16 stadi acquisisce due ingressi di 32 bit e produce due uscite di 32 bit; l'output a sinistra è semplicemente una copia dell'input a destra, mentre l'output a destra è l'OR esclusivo bit a bit dell'input a sinistra e di una funzione dell'input a destra e della chiave per questo stadio. Tutta la complessità è racchiusa in questa funzione.

C'è da notare come tale algoritmo così com'è possa essere utilizzato solo per cifrare blocchi di testo in chiaro di 64 bit. A seconda della strategia adottata per cifrare mediante il DES blocchi di dimensione più elevata si parla di **DES Modes of Operation** (Modi di Operazione DES).

Il metodo più semplice potrebbe essere quello di dividere il testo in blocchi di 64 bit, cifrarli tutti con la stessa chiave e spedirli sequenzialmente. Tale procedimento va sotto il nome di Modo di Operazione **ECB** (*Electronic CodeBook*).

Poiché tale procedimento è poco sicuro e rende facile ad un possibile intruso collezionare faclmente una grande quantità di messaggi cifrati con la stessa chiave, è stato

ideato il Modo di Operazione **CBC** (*Cipher Block Chaining*), con cui ogni nuovo blocco di 64 bit di testo in chiaro, prima di essere cifrato, viene sottoposto ad una operazione di XOR con i precedenti 64 bit ottenuti dalla cifratura. L'unico accorgimento aggiuntivo, in tal caso, dovrà essere quello di stabilire un *Vettore di Inizializzazione* (*IV, Initialization Vector*) che possa essere utilizzato per effettuare l'operazione di XOR per il primo blocco. Tale vettore di inizializzazione dovrà essere noto ad entrambe le parti interessate alla comunicazione e dovrà essere mantenuto segreto (ad esempio comunicandolo con le stesse modalità della chiave da adoperare).

Per applicazioni specifiche sono stati sviluppati altri modi di operazione più complessi, come l'**OFB** (*Output FeedBack*) ed il **CFB** (*Cipher FeedBack*) che richiedono l'utilizzo di uno shift-register, ma sono poco utilizzati nella pratica in quanto il modo CBC si dimostra già sufficientemente affidabile.

4.8 L'algoritmo RSA

Come detto, l'algoritmo RSA prende il nome dai suoi tre ideatori, Rivest, Shamir e Adleman [FIRE]. L'algoritmo consiste di tre fasi separate:

1. la generazione della coppia di chiavi pubblica/privata;
2. la cifratura del testo in chiaro;
3. la decifrazione del messaggio.

Per ottenere le chiavi bisogna generare due numeri primi sufficientemente grandi (preferibilmente dello stesso ordine di grandezza). Detti 'p' e 'q' questi due numeri, si definirà come 'n' il loro prodotto $n=pq$, come 'd' un qualunque numero che sia primo rispetto a $(p-$

1)(q-1), ed infine come ‘e’ l’inverso moltiplicativo di ‘d’ con modulo (p-1)(q-1).

Fatto ciò, i numeri p e q potranno essere dimenticati (ma non rivelati), mentre la coppia (n,e) costituirà la chiave pubblica dell’algoritmo e la coppia (n,d) la chiave privata. C’è da notare che ‘d’ ed ‘e’ sono intercambiabili, in quanto sono uno l’inverso dell’altro, e dunque potrebbero anche essere scambiati.

Tutto ciò di cui consistono la cifratura e la decifrazione non sono altro che delle semplici operazioni di elevazione a potenza in modulo; ovvero, il numero da cifrare (il messaggio in chiaro deve essere opportunamente convertito in una sequenza di numeri) dovrà essere elevato ad ‘e’ in modulo ‘n’, ed il numero ottenuto sarà il testo cifrato che potrà essere trasmesso sulla rete. Viceversa la decifrazione consisterà nella elevazione alla potenza ‘d’ con modulo ‘n’ del numero cifrato, in modo da riottenere il numero di partenza.

Tale algoritmo si basa interamente sulla difficoltà di fattorizzazione di numeri molto elevati, in tal caso del numero ‘n’ ottenuto come prodotto di due numeri primi molto grandi; se infatti si riuscisse (in un tempo ragionevole) a fattorizzare tale numero, trasmesso in chiaro sulla rete, si potrebbe facilmente ricavare ‘d’ da ‘e’ ed il messaggio sarebbe decifrato.

Anche se l’algoritmo in apparenza è molto più semplice di altri algoritmi (come ad esempio il DES), in realtà presenta notevoli difficoltà di implementazione che lo rendono anche di applicazione abbastanza lenta, il che ha spinto molti produttori a realizzare delle versioni hardware dell’algoritmo RSA su chip dedicati molto veloci. I vari problemi si presentano sia per la generazione di numeri primi di elevata grandezza, sia per trovare gli inversi moltiplicativi, e sia, infine, per eseguire correttamente le operazioni di elevazione a potenza in modulo senza generare errori di overflow sulla macchina utilizzata.

Fortunatamente diversi strumenti matematici implementabili in software tornano utili per ognuno dei singoli problemi appena esposti. Ad esempio, per la generazione di numeri primi di elevata grandezza, ci si limiterà a generare dei numeri elevati e a sottoporli ad

opportuni *test di primalità* per verificare se siano o meno primi, continuando finché non si trovi un numero che soddisfi tale condizione. Infatti, per numeri molto elevati, non si può mai avere la certezza che un numero sia effettivamente primo, e ci si limiterà allora a prendere dei numeri che abbiano una elevata probabilità di esserlo (ma non la certezza). Il test di primalità più conosciuto e più diffuso è il **test di Rabin-Miller** [SCHN].

Per quanto riguarda invece il calcolo degli inversi moltiplicativi, si dimostra di grande utilità l'**Algoritmo di Euclide Esteso** [STAL], che permette di trovare sia il massimo divisore comune tra due numeri interi e sia, se esiste, anche l'inverso moltiplicativo. Infine, per l'elevazione a potenza in modulo, bisogna solo aver cura di convertire l'elevazione in una serie di prodotti sempre dello stesso numero, avendo cura ad ogni passaggio intermedio di effettuare l'operazione di modulo sul risultato.

Come detto, tutti questi strumenti possono essere implementati in software ma si dimostrano abbastanza lenti, ma per fortuna gli algoritmi a chiave pubblica non vengono solitamente utilizzati per condurre tutta una conversazione tra due utenti remoti ma solo per lo scambio della chiave da impiegare per il successivo algoritmo simmetrico.

4.9 Autenticazione in UNIX

UNIX prevede una forma di autenticazione tramite password che sarà ora descritta. Il processo di login chiede all'utente di digitare il proprio nome di login e la propria password. La password inserita viene immediatamente cifrata [TAN1, STEV].

Il processo legge poi il file delle password (`/etc/passwd`) che contiene una serie di linee ASCII, una per ogni utente, che hanno il seguente formato costituito da 7 campi separati l'uno dall'altro da un segno di due punti:

<login_name>:<cipher_passwd>:<UID>:<GID>:<misc>:<login_dir>:<shell>

Un esempio di riga del file delle password potrebbe essere il seguente:

`ffazzino:9A8B7C6D5E4F0312:1043:24:FabrizioFazzino:/home/ffazzino:/bin/bash`

Il processo legge tale file finché trova il *login-name* (o *user-name*) che l'utente ha digitato quando si è collegato al sistema, così viene effettuato il login solo se la password (cifrata) contenuta nella riga corrispondente è uguale a quella appena digitata.

In UNIX l'autenticazione è necessaria in quanto il sistema operativo tiene nota di quali sono i proprietari di ciascun file, e tutti i file possono essere protetti dagli utenti non autorizzati agendo sugli appositi **bit dei diritti** (*rights bits*). Tali bit, nove in tutto, controllano se autorizzare o meno la lettura, la scrittura e l'esecuzione del file da parte rispettivamente del proprietario del file, degli altri membri del gruppo e di tutte le altre persone.

4.10 Sicurezza sul Web

Per quanto riguarda la sicurezza di un utente del World Wide Web, tutti i più recenti browser permettono di limitare e rendere più sicure le comunicazioni con il Web server da cui provengono.

In particolare, per molte applicazioni, specialmente di vendita a distanza tramite Internet, all'utente connesso col proprio browser viene chiesto di digitare delle informazioni che si desidera mantenere riservate, come ad esempio il numero della carta di credito od una parola d'ordine, per cui inviarle in chiaro sulla rete potrebbe essere estremamente pericoloso.

Per garantire la riservatezza delle informazioni trasmesse tra il Web server ed il browser è stata inizialmente studiata una estensione del protocollo HTTP, detta **S-HTTP** (*Secure*

HTTP), che permetteva la trasmissione di pagine Web autenticate e cifrate, consentendo lo scambio su Internet di informazioni protette.

In seguito, considerate le limitate garanzie di riservatezza e di sicurezza offerte dall'ormai invecchiata implementazione del protocollo S-HTTP, la società Netscape ha sviluppato un nuovo protocollo per garantire un livello di sicurezza elevato alle transazioni via Internet e per favorire la crescita del commercio elettronico.

Tale nuovo protocollo, detto **SSL** (*Secure Sockets Layer*), prevede l'istituzione di un canale indipendente di comunicazione, trasportato da TCP/IP, che gli permette di operare in maniera trasparente con qualunque altro protocollo come HTTP o FTP. Tale protocollo, basato sullo scambio di chiavi a 40 bit, e che si è ulteriormente evoluto con le versioni SSL 2.0 e SSL 3.0, offre un grado di sicurezza molto maggiore di quello offerto da S-HTTP ed è supportato da tutti i più recenti browser.

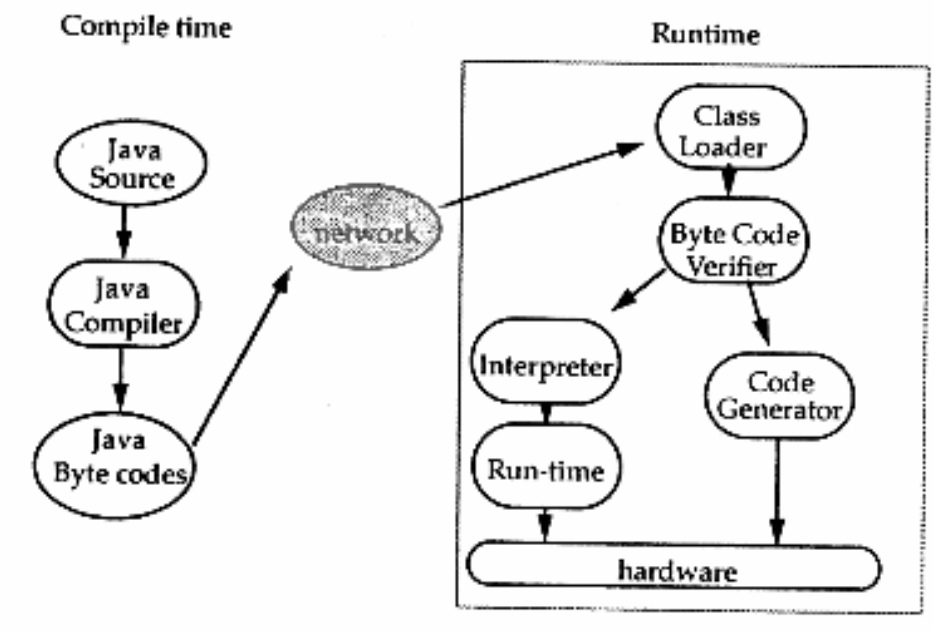
Rimane solo da osservare che questi protocolli agiscono ad un livello molto più basso di quello gestibile dallo sviluppatore comune (e tanto meno dall'utente occasionale), in quanto ad occuparsene è la coppia Web server / Web browser; allora l'utente se ne può servire solo se sia il proprio Web browser quanto il Web server remoto dispongono della relativa opzione. Mentre però per i browser le innovazioni vengono prontamente incorporate nelle nuove versioni, che vengono poi spesso distribuite sotto forma di freeware o shareware, la strategia dei produttori è invece quella di mantenere il costo dei Web server tipicamente elevato, rendendo difficile un continuo aggiornamento tecnologico.

All'utente collegato al browser non rimane che verificare se la pagina con cui è collegato è sicura o meno (ad esempio il browser più diffuso, Netscape Navigator, presenta sulla barra di stato il disegno di una chiave spezzata se il Web server da cui la pagina è stata scaricata non prevede protocolli di trasferimento sicuri, mentre il disegno diventa di una chiave intera solo se il Web server dispone della relativa opzione ed essa è stata attivata).

4.11 Sicurezza con Java

La macchina virtuale di Java pone delle serie restrizioni alla modalità con cui applicazioni ed applet possono essere eseguite; in particolare esiste un modulo, detto *Byte Code Verifier*, che al momento dell'esecuzione verifica che il codice da eseguire non contenga delle istruzioni fraudolente, che tentino di scavalcare i limiti imposti dal linguaggio (ad esempio tentando di accedere direttamente alla memoria).

Sarebbe infatti possibile, studiando gli *opcode* della macchina virtuale di Java, scivere 'a mano' un programma che tenti ad esempio un accesso illecito alla memoria non occupata dall'applicazione (o applet) in esecuzione, senza passare dalla compilazione, utilizzando una sorta di linguaggio assembly.



Il Byte Code Verifier sopperisce a tutto questo e garantisce che il codice da eseguire non possa compiere alcun tipo di istruzione illecita [PROG], ovvero che non vengano simulati puntatori alla memoria, che non vengano violate le restrizioni di accesso e che gli oggetti

vengano utilizzati sempre per quello che sono: in tal modo non esiste il rischio di overflow o underflow dello stack, di parametri non corretti o di accessi illegali da parte degli oggetti.

Il processo di verifica interessa anche quelle classi che vengono importate direttamente dalla rete durante l'esecuzione di una applet. Inoltre, per quanto riguarda in particolare le applet, per evitare attacchi all'utente nascosti all'interno di una applet scaricata da Internet i Web browser Java-enabled pongono delle serie restrizioni alle modalità con cui una applet può essere eseguita in locale. Infatti senza che l'utente lo sappia una applet potrebbe, oltre alle sue normali evidenti funzionalità, leggere un file riservato dal disco dell'utente e spedirlo ad un host estraneo (si tratterebbe in questo caso di un *trojan horse*), oppure inondare di file il disco dell'utente fino a bloccargli il sistema.

Allora, per evitare questi attacchi, i Web browser Java-enabled pongono per le applet, siano esse scaricate dalla rete o anche eseguite in locale, le seguenti restrizioni [JTUT]:

- le applet non possono definire dei metodi nativi in linguaggio C;
- le applet non possono accedere al file-system dell'host che le esegue, per cui non possono leggere i file dell'utente, né modificarli, né crearne di nuovi;
- le applet non possono avviare l'esecuzione di altri programmi (mediante il metodo *exec*) sul sistema ospite;
- le applet non possono accedere direttamente alla memoria, anche grazie all'eliminazione dei puntatori, per cui possono solo definire degli oggetti e lavorare con essi (questa caratteristica vale anche per le applicazioni stand-alone);
- le applet non possono leggere nessuna proprietà del sistema ospite;
- le finestre che le applet presentano all'utente sono diverse da quelle delle applicazioni stand-alone (ad esempio presentano una piccola icona caratteristica), in modo da permettere all'utente di distinguerle dalle applicazioni sicure;

- le uniche comunicazioni delle applet con l'esterno possono avvenire tramite la rete mediante socket, ma soltanto con l'host da cui le applet sono state scaricate.

Queste restrizioni rendono l'utente praticamente sicuro che l'applet non potrà in alcun modo carpirgli informazioni o danneggiare il suo sistema; infatti, come evidenziato in [JSEC], l'unica possibilità di attacco da parte di una applet rimane quella di espandersi in memoria (allocandola dinamicamente) fino ad occupare tutta quella che il sistema operativo locale gli alloca, ma tale attacco non è di per sé pericoloso in quanto tutti i moderni sistemi operativi multi-tasking permettono all'utente di terminare una applicazione qualora sia evidente che stia rendendo instabile il sistema.

Da notare che se l'applet necessita di manipolare dei file su disco o di avviare l'esecuzione di un processo, l'unica soluzione percorribile rimane quella di implementare un modello Client/Server, in cui l'applet costituisce il lato client, e sarà il lato server ad occuparsi di leggere o scivere su disco o di eseguire altri processi [JTUT]. Per quanto detto, tale server deve per forza trovarsi sull'host da cui la applet è stata scaricata, e per poter eseguire le operazioni in questione non può trattarsi di un'altra applet ma deve essere una applicazione Java stand-alone.

5 - Modelli di Sicurezza

5.1 Il modello di Bell - LaPadula (BLP)

Un **modello di sicurezza** è un modello concettuale di alto livello, indipendente dal software, che parte dalle specifiche dei requisiti di protezione del sistema per descrivere le proprietà funzionali e strutturali delle politiche di sicurezza adottate. [DBSE]

Tale modello, oltre a permettere agli sviluppatori di fornire una definizione di alto livello dei requisiti di protezione e delle politiche del sistema, consente di produrre una concisa descrizione formale del comportamento del sistema e di dimostrare che il sistema di sicurezza soddisfa alcune proprietà.

Il più importante modello di sicurezza, che sarà anche adoperato, opportunamente riadattato, per il Laboratorio Virtuale, è il **modello di Bell e LaPadula (BLP)**.

Il modello BLP, definito negli anni tra il 1973 e il 1976, è un modello applicabile ai Sistemi Operativi e alle Basi di Dati, ed è basato sul *paradigma Soggetto-Oggetto*, ovvero le varie entità sono catalogate in entità attive (gli elementi attivi del sistema che possono eseguire azioni, ovvero gli utenti o i loro processi) ed entità passive (gli elementi del sistema che contengono le informazioni e verso cui bisogna controllare l'accesso).

Il modello BLP associa ad ogni entità un **livello di sicurezza** (*security level*) costituito da una classificazione (*classification*) gerarchica (`TopSecret > Secret > Confidential > Unclassified`) e da un insieme di categorie (*set of categories*) non

gerarchico che indica il settore di appartenenza (es. posta elettronica, prodotti commerciali, centrali atomiche ecc.).

Ogni soggetto può inoltre accedere ai vari oggetti secondo vari **modi di accesso** (*access modes*), detti *Read-only*, *Append*, *Execute* e *Read-write*, ripettivamente per la sola lettura, per l'aggiunta (senza lettura), l'esecuzione (per i programmi) e la lettura-scrittura. Più concisamente i quattro modi di accesso vengono indicati *read*, *append*, *execute* e *write*.

5.2 Le strutture dati del modello BLP

Il modello BLP richiede che lo stato del sistema venga memorizzato in apposite strutture dati, in modo che in ogni momento tale stato possa essere descritto dalla quartupla `(b,M,f,H)`.

In particolare tali strutture sono:

- **b (*Current Access Set*):** è composto da triple della forma `<s, o, m>`, che indicano che il soggetto 's' ha attualmente accesso all'oggetto 'o' nel modo di accesso 'm'.
- **M (*Access Matrix*):** la matrice degli accessi `M(s,o)` indica il modo di accesso autorizzato per il soggetto 's' sull'oggetto 'o'.
- **f (*Level Function*):** è una funzione che associa ad ogni soggetto od oggetto nel sistema un livello di sicurezza. Agli oggetti è associato solo un livello di sicurezza; invece ai soggetti (ovvero agli utenti) sono assegnati un livello iniziale (detto *clearance*) ed un livello attuale (*current level*) che può anche cambiare durante il tempo di vita del soggetto pur non potendo mai superare la clearance.
- **H (*Current Objects Hierarchy*):** la gerarchia degli oggetti attuali è un albero diretto i cui nodi corrispondono ad oggetti nel sistema. Il modello richiede che la gerarchia soddisfi la

proprietà di compatibilità, che stabilisce che il livello di sicurezza di un oggetto deve dominare quello del suo genitore.

5.3 Operazioni nel modello BLP

Lo stato del sistema può cambiare eseguendo delle operazioni. Le operazioni consentite e le relative transizioni di stato sono le seguenti:

- **Get access:** quando un soggetto guadagna l'accesso ad un oggetto bisogna aggiungere in 'b' la corrispondente terna.
- **Release access:** quando il soggetto termina l'accesso bisogna rimuovere da 'b' la tripla corrispondente. E' l'operazione inversa alla precedente.
- **Give access:** permette che i diritti di accesso ad un oggetto possano essere propagati da un soggetto ad un altro; bisogna allora inserire nell'opportuno elemento della matrice degli accessi 'M' il nuovo modo di accesso consentito. L'operazione viene eseguita solo se rispetta le politiche di sicurezza.
- **Rescind access:** revoca un accesso precedentemente garantito con l'operazione precedente, eliminando da 'M' il corrispondente modo di accesso; inoltre se quel permesso sta venendo attualmente sfruttato bisogna forzare il rilascio dell'oggetto cancellando la opportuna terna da 'b'. Affinché questa operazione possa essere eseguita, il soggetto richiedente deve avere il permesso di scrittura sul genitore dell'oggetto interessato nella gerarchia di oggetti 'H'.
- **Create object:** attiva un oggetto inattivo rendendolo accessibile, aggiungendo alla gerarchia di oggetti 'H' il nodo corrispondente all'oggetto attivato.
- **Delete object:** disattiva un oggetto attivo rimuovendo da 'H' il nodo corrispondente e tutti i nodi da esso discendenti; inoltre se l'oggetto stava venendo utilizzato bisogna anche

forzarne il rilascio cancellando la tripla opportuna in ‘b’. E’ l’operazione inversa alla precedente.

- **Change subject security level:** cambia l’attuale livello di sicurezza ‘f’ di un soggetto modificando la corrispondente funzione. Il nuovo livello deve essere inferiore della clearance attribuita inizialmente al soggetto.
- **Change object security level:** ha l’effetto di attribuire un nuovo valore alla funzione ‘f’ dell’oggetto inattivo; tale livello può solo essere aumentato, e non deve mai superare la clearance del soggetto richiedente il cambiamento.

5.4 Proprietà del modello BLP

Il modello BLP definisce un insieme di proprietà che devono essere soddisfatte affinché il sistema possa essere definito sicuro: uno stato del sistema è sicuro se, e solo se, sono soddisfatte le proprietà.

Ogni operazione richiesta è controllata da un *reference monitor*, e la sua esecuzione è concessa se, e solo se, lo stato risultante in cui si verrebbe a trovare il sistema sia sicuro, ovvero soddisfi tutte le proprietà del modello.

Le proprietà da soddisfare per la sicurezza dello stato del sistema sono le seguenti:

1. **proprietà di Simple Security** (*SS-property*): un soggetto può accedere ad un oggetto per una operazione di read o write solo se la sua clearance è maggiore od uguale a quella dell’oggetto;
2. **proprietà Star** (**-property*): un soggetto può avere un accesso differente all’oggetto a seconda del suo attuale livello di sicurezza rispetto a quello dell’oggetto; ovvero può accedere solo per operazioni di append se ha un livello inferiore, per operazioni di write se

ha un livello uguale e per operazioni di read se ha un attuale livello di sicurezza superiore a quello dell'oggetto.

3. **proprietà di Discretionary Security** (*DS-property*): ogni soggetto può esercitare solo gli accessi per cui ha la necessaria autorizzazione, cioé uno stato soddisfa tale proprietà se e solo se per tutti gli accessi correnti `<s,o,m>` di 'b' il modo di accesso 'm' è presente nel corrispondente elemento della matrice degli accessi `M[s,o]`.

Le prime due proprietà insieme servono a garantire la sicurezza delle informazioni, e vanno sintetizzate con la dizione *"No Read-Up / No Write-Down Secrecy"* : cioé non è possibile leggere informazioni ad un livello di sicurezza superiore al proprio (come è ovvio), ma neanche scrivere informazioni ad un livello inferiore.

Queste regole insieme servono ad evitare che le informazioni possano propagarsi senza controllo da parte del proprietario, e sono lo strumento per impedire che il sistema possa essere espugnato da un trojan horse.

6 - I Sistemi Distribuiti

6.1 Obiettivi

Dopo avere brevemente tratteggiato gli argomenti che saranno utili per la realizzazione del Laboratorio Virtuale (il Web, Java, la security, la crittografia, i modelli di sicurezza), è possibile ora ritornare a quanto detto nel primo capitolo a proposito del VL ed approfondire la sua definizione.

Abbiamo già visto come in definitiva il laboratorio debba offrire agli utenti delle risorse di vario genere (ad esempio impianti o simulatori) accessibili mediante delle interfacce, in modo da poter interagire con esse in vari modi (precostituiti o da assemblare) con le finalità più svariate, sia didattiche (*remote tutoring*) che di ricerca o di tele-lavoro (*remote control, monitoring & performance evaluation*).

Per consentire tali interazioni, il Laboratorio Virtuale deve essere progettato come un Sistema Distribuito, ovvero un sistema composto da una collezione di macchine che non hanno memoria condivisa ma appaiono tuttavia agli utenti come se fossero un solo calcolatore. In particolare, come vedremo adesso, ci si riferisce ad un Sistema Distribuito come ad un particolare tipo di sistema dotato di CPU multiple [TANE].

6.2 Classificazione dei sistemi con CPU multiple

I calcolatori con più CPU, ed in particolare quelli con flussi multipli di istruzioni e di dati, possono essere catalogati in base al fatto che l'**hardware** consenta o meno la condivisione della memoria:

1. **Multiprocessori**: sono calcolatori con memoria condivisa, e tendono ad essere *ad accoppiamento stretto*, ovvero la trasmissione tra due calcolatori è veloce; sono tipicamente usati come sistemi paralleli di calcolo.
2. **Multicalcolatori**: sono calcolatori con memoria privata non condivisa, e tendono ad essere *ad accoppiamento lasco*, ovvero la trasmissione tra due calcolatori è abbastanza lenta, come quella che si può avere su una rete geografica; sono tipicamente usati come sistemi distribuiti.

Gli stessi sistemi con CPU multiple possono anche essere classificati in base al **software** che essi utilizzano:

1. **Software ad accoppiamento lasco**: le macchine sono tutte tra loro indipendenti, e solo quando è necessario interagiscono tra loro in maniera limitata (secondo quello che avviene ad esempio in una rete locale di computer).
2. **Software ad accoppiamento stretto**: le macchine interagiscono fra loro per svolgere quasi tutte le operazioni (come ad esempio i Sistemi Operativi e gli applicativi adoperati per l'elaborazione parallela).

In base alla catalogazione hardware e software dei sistemi con CPU multiple possono presentarsi quattro casi; in particolare, tra questi vengono detti **Sistemi Distribuiti** quelli in cui si adopera un software ad accoppiamento stretto su di un hardware ad accoppiamento lasco.

Un Sistema Distribuito gira dunque su una collezione di macchine che sebbene non abbiano memoria condivisa appaiono agli utenti come un singolo calcolatore; le macchine

connesse dalla rete lavorano come un **uniprocessore virtuale** e l'utente può non sapere che ci sono molte CPU.

La caratteristica principale di un Sistema Distribuito deve essere un **meccanismo di comunicazione** tra processi identico su tutte le macchine e senza distinzioni tra la comunicazione locale e quella remota; le varie macchine comunicano dunque tra loro tramite messaggi previo accordo sui protocolli di rete da adoperare.

6.3 Vantaggi dei Sistemi Distribuiti

Un Sistema Distribuito presenta rispetto ad un Sistema Centralizzato i seguenti vantaggi:

1. **Economicità**: mentre fino a 30 anni fa, in accordo con la cosiddetta Legge di Grosch [TANE], la potenza di calcolo di una CPU era proporzionale al quadrato del suo prezzo, e allora tutte le società acquistavano la macchina più grande che potevano permettersi, oggi questo non è più vero ed anzi un sistema distribuito di tante CPU offre un migliore rapporto qualità/prezzo rispetto ai mainframe isolati.
2. **Velocità**: la potenza di calcolo delle CPU in un sistema distribuito si somma, raggiungendo velocità che in un solo processore di mainframe sono impossibili da raggiungere; ad esempio 1000 CPU da 20 MIPS (*Millions Instructions Per Second*, milioni di istruzioni al secondo) garantiscono una potenza di elaborazione di 20000 MIPS.
3. **Distribuzione intrinseca**: alcune applicazioni sono intrinsecamente distribuite, come ad esempio le operazioni compiute dalle banche o nelle fabbriche.
4. **Affidabilità**: se una o più macchine si guastano, il resto del sistema continua ugualmente a funzionare.

5. **Crescita incrementale**: mentre se si acquista un mainframe e diventa insufficiente si è perso il capitale investito, un Sistema Distribuito può essere espanso gradualmente semplicemente aggiungendo processori.
6. **Condivisione di risorse**: un Sistema Distribuito permette a molti utenti di condividere le stesse risorse, ad esempio basi di dati o periferiche.
7. **Efficienza**: un Sistema Distribuito può ripartire il carico di lavoro sulle macchine disponibili in modo molto efficiente, in modo che ogni lavoro possa essere eseguito sulla macchina più appropriata.

6.4 Scelte di progetto nei Sistemi Distribuiti

Uno dei principali requisiti che si vuole imporre ad un Sistema Distribuito è la **trasparenza**: si dice trasparente un sistema che fa sembrare che una collezione di macchine sia un sistema uniprocessore, nascondendo la distribuzione sia ad alto livello agli utenti, sia magari anche a basso livello ai programmatori, progettando l'interfaccia delle chiamate di sistema in modo da mascherare l'esistenza di processori multipli.

Un'altra caratteristica richiesta è la **flessibilità**, in modo tale che se certe scelte di progetto si dimostrano in seguito sbagliate bisogna poi poter ritornare sui propri passi. Per ottenere una buona flessibilità, un Sistema Distribuito non dovrà essere implementato con un **Modello a Kernel Monolitico**, in cui vi è un unico nucleo che provvede a fornire la maggior parte dei servizi, bensì con un **Modello a Microkernel**, in cui il kernel provvede solo al minimo indispensabile ed il grosso dei servizi offerti vengono resi disponibili sotto forma di server che girano a livello utente.

In tal modo l'utente per ottenere qualche servizio non compreso nel Microkernel deve inviare un messaggio al server appropriato, utilizzando l'interfaccia ben definita per ognuno dei servizi. Tale modello a Microkernel è molto flessibile perché si possono sempre aggiungere nuovi servizi senza arrestare il sistema, quando invece un kernel monolitico dovrebbe essere interamente sostituito.

Un altro requisito che si desidera da un Sistema Distribuito è l'**affidabilità**, ovvero si vorrà fare in modo che se una macchina va fuori servizio i suoi client non cadano con essa, ma ce ne sia un'altra che si prenda cura di svolgere il suo lavoro, ridondando parti software ed hardware; si parla in tal caso di *fault tolerance* (tolleranza ai guasti). Purtroppo però nella pratica nei Sistemi Distribuiti alcuni tipi di server sono indispensabili e non possono essere ridondati.

Un aspetto strettamente correlato è quello della **sicurezza**, che rispetto ai sistemi uniprocessore è più critica in quanto se ad un server arriva un messaggio bisogna potersi assicurare dell'identità del mittente.

Altre caratteristiche dei Sistemi Distribuiti riguardano le **prestazioni**: poiché avviare attività in parallelo su vari processori comporta la spedizione di molti messaggi sulla rete, su cui le comunicazioni sono relativamente lente, bisogna allora ripartire opportunamente in parallelo sulle varie macchine le varie operazioni prestando attenzione alla *grana di parallelismo delle computazioni*. Infatti le singole operazioni non devono essere né troppo impegnative, altrimenti sfrutto poco il parallelismo, né troppo poco, altrimenti il ritardo delle comunicazioni incide troppo sui piccoli tempi di computazione.

L'ultimo requisito richiesto ad un Sistema Distribuito è la **scalabilità**, ovvero deve poter funzionare ugualmente bene anche se il numero di macchine che ne fanno parte aumenta a dismisura. A tal fine bisognerà evitare tutti i componenti centralizzati, come ad esempio un unico server utilizzato per tutti i client, in quanto tali componenti rappresentano un collo di

bottiglia. Essi infatti richiederebbero l'impiego bande passanti enormi sulla rete, e sarebbero molto vulnerabili, in quanto il guasto di un dispositivo centralizzato porterebbe al blocco tutto il sistema.

6.5 Modalità di comunicazione nei Sistemi Distribuiti

La modalità di comunicazione tra processi è la differenza più importante dei Sistemi Distribuiti dai sistemi uniprocessore, in quanto nei Sistemi Distribuiti non vi è memoria condivisa tra le varie macchine.

Gli esempi principali di modalità di comunicazione tra due processi in un Sistema Distribuito sono i protocolli, i modelli Client/Server e la chiamata di procedura remota.

Poiché non vi è memoria condivisa, tutte le comunicazioni devono basarsi sullo scambio di messaggi lungo la rete; viene detto a tal proposito **protocollo** l'insieme delle regole che definiscono il formato, i contenuti ed il significato dei messaggi. Esempi di protocolli sono OSI (*Open Systems Interconnection*) ed il TCP/IP adoperato su Internet.

A loro volta i protocolli si dicono *Connection-Oriented* (orientati alla connessione), se prima di scambiarsi i dati mittente e destinatario stabiliscono esplicitamente una connessione e la richiudono alla fine della comunicazione, oppure *Connectionless* nel caso in cui ogni messaggio viene spedito indipendentemente dagli altri senza prendere accordi.

Purtroppo se in un Sistema Distribuito tutte le comunicazioni fossero basate solo sui protocolli sarebbe estremamente oneroso gestire i singoli messaggi, ed in più i vari protocolli indicano solo come trasmettere dei bit da un mittente ad un destinatario senza specificare come debba essere strutturato il Sistema Distribuito.

Si utilizzano allora i **modelli Client/Server**, in cui vi è un insieme di processi cooperanti, detti *server*, che offrono servizi agli utenti, detti *client*. Di solito in un modello Client/Server tutte le macchine fanno girare lo stesso tipo di Microkernel mentre client e server girano come processi utente.

Di solito i modelli Client/Server evitano l'overhead dei protocolli orientati alla connessione, come OSI ed il TCP/IP, e si basano su di un semplice *protocollo Request/Reply* (richiesta/risposta) privo di connessione. Così ogni client può spedire un messaggio al server chiedendo qualche servizio ed il server risponde restituendo i dati richiesti od un codice di errore.

I vantaggi principali dell'utilizzo di un tale modello sono la semplicità, in quanto non vi è la necessità di stabilire la connessione e la risposta del server serve anche come accettazione della richiesta, e l'efficienza, perché le uniche operazioni da compiere sono quelle richieste dal protocollo Request/Reply che definisce l'insieme di richieste e risposte.

In definitiva per un Sistema Distribuito organizzato secondo un modello Client/Server tutti i servizi offerti dal Microkernel possono essere ricondotti a due sole chiamate di sistema, ovvero `send` e `receive`. Solitamente tali primitive per lo scambio di messaggi sono primitive bloccanti, ovvero i processi che le chiamano rimangono bloccati in attesa che venga completata l'operazione richiesta.

Purtroppo però strutturare un Sistema Distribuito secondo un modello Client/Server lo rende totalmente dissimile da un Sistema Centralizzato, e così si perde la caratteristica della trasparenza, ovvero per l'utente tale sistema non si comporta più come un uniprocessore virtuale.

Si introduce allora la **chiamata di procedura remota** (RPC, *Remote Procedure Call*) con cui si rende possibile ad un programma chiamare procedure allocate su di un'altra macchina, che vengono dunque eseguite su una macchina remota. Le uniche informazioni che il

chiamante e la procedura chiamata possono scambiarsi sono i parametri della procedura ed il risultato, ed il programmatore non ha altre visibilità né di scambio messaggi né di I/O.

Allora utilizzando la RPC l'unico problema di cui deve occuparsi il programmatore è il passaggio dei parametri, che dovranno essere opportunamente impacchettati nella chiamata secondo il *parameter marshalling* (inquadramento dei parametri) previsto per quella procedura. Tale problema del formato dei parametri si pone soprattutto nel caso in cui per implementare la RPC si utilizzi un linguaggio di programmazione con una tipizzazione debole, come il C, piuttosto che uno con una tipizzazione forte come il Pascal e Java. Infatti il C lascia una grande libertà di scelta sulla dimensione, il tipo ed il numero dei parametri di ogni funzione, ed il programmatore dovrà stare attento ad utilizzare per ogni chiamata il corretto tipo e numero di parametri, che sono noti.

Oltre al formato il programmatore dovrà anche curare le modalità di rappresentazione dell'informazione, in quanto ogni macchina utilizza una propria rappresentazione interna sia per i valori numerici (es. little-endian o big-endian) che per i caratteri (es. codici ASCII o EBCDIC).

In realtà, a parte la visibilità che ne ha il programmatore, anche le RPC vengono implementate a basso livello mediante lo scambio di messaggi sulla rete; allora tra le scelte implementative può esservi quella di utilizzare un protocollo con o senza connessione, ed in particolare uno già esistente od uno appositamente dedicato a questo scopo.

6.6 Processori e modelli di sistema

In un Sistema Distribuito i processori possono essere normalmente organizzati in due modi opposti, anche se spesso si adottano delle soluzioni ibride tra le due che saranno proposte.

Con il **modello Workstation** il sistema consiste di più personal computer di fascia alta fisicamente distribuiti all'interno di un edificio o di un campus e connessi da una rete locale. In ogni istante ogni workstation ha un solo utente collegato oppure è inattiva, dunque il possessore ha a disposizione una quantità fissa di potenza di calcolo ed ha un tempo di risposta garantito.

Lo svantaggio principale di questo approccio è la scarsa flessibilità, in quanto ogni utente ha sempre a disposizione una sola CPU, anche quando non la usa e quando avrebbe bisogno di risorse di calcolo superiori; si potrebbe allora fare in modo, se si riuscisse a diagnosticare quali workstation sono inattive, che queste venissero utilizzate per l'esecuzione remota di processi provenienti da altre macchine.

Una workstation può essere definita inattiva se non esegue processi utente (ma ovviamente esegue i vari demoni di posta, clock, ecc.) e non ha ricevuto un input da tastiera da molto tempo. Gli algoritmi per individuarle possono partire dalla macchina inattiva, che può prendere l'iniziativa e comunicarlo alle altre macchine mediante un registro centralizzato od inviando messaggi in broadcast, oppure possono partire dal client che vuole eseguire un programma in remoto e lo comunica in broadcast attendendo le risposte delle workstation inattive dotate delle caratteristiche necessarie per l'esecuzione remota del processo.

Invece con il **modello a Pool di processori** il sistema consiste di un insieme coordinato di CPU che possono essere allocate agli utenti dinamicamente su richiesta. In tale modello gli utenti non hanno a disposizione delle workstation personali ma semplicemente dei terminali, e tutte le CPU è come se fossero delle workstation inattive a cui si può accedere dinamicamente a seconda delle richieste di esecuzione di processi utente.

Con tale modello si possono raggiungere potenze di calcolo impossibili per un solo processore, e sono ottimi anche il rapporto prestazioni/prezzo, l'affidabilità e la tolleranza ai guasti. Inoltre non vi è bisogno di verificare continuamente se vi siano workstation inattive e non vi è nulla da trasmettere alla macchina personale dell'utente, in quanto tutti gli accessi avvengono tramite terminali connessi direttamente al sistema.

Per sistemi che devono essere impegnati pesantemente è allora molto più semplice e potente realizzare un pool di processori, quando invece un utente amatoriale potrà accontentarsi di utilizzare una workstation personale. Un **modello ibrido** potrebbe essere implementato come compromesso, fornendo ad ogni utente una workstation personale ed avendo nel frattempo un pool di processori, in modo da svolgere ad esempio sulle workstation il lavoro interattivo (che godrebbe così di un tempo di risposta garantito) e non utilizzando per semplicità le workstation inattive, ma eseguendo sul pool di processori i processi non interattivi come tutte le computazioni pesanti. In tal modo si potrà ottenere una veloce risposta interattiva, una efficiente utilizzazione delle risorse ed una semplice fase di progetto.

6.7 Algoritmi di allocazione dei processori

Per definizione in un Sistema Distribuito sono presenti più processori, e bisogna allora sviluppare degli **algoritmi** per allocare i processi ai vari processori. In particolare, per il modello a Workstation bisogna decidere se far girare il processo in locale oppure cercare una workstation inattiva, mentre per il modello a Pool di processori bisogna in ogni caso decidere per ogni nuovo processo a quale processore debba essere allocato.

Supponendo che tutte le CPU siano equivalenti per l'esecuzione dei processi, quando viene creato un nuovo processo (ad esempio dalla shell in risposta al comando dell'utente)

l'algoritmo di allocazione dovrà fare in modo da scegliere la CPU più appropriata per la sua esecuzione; con questo si vuole indicare che ogni algoritmo deve avere un **obiettivo da ottimizzare**, ad esempio rendendo massima l'*utilizzazione delle CPU* (cioé il numero di cicli effettivamente eseguiti in favore di processi utente rispetto ai cicli totali in cui la CPU può anche essere inattiva), minimizzando il *tempo di risposta* (cioé il tempo impiegato ad eseguire il processo) oppure, con una piccola variante, rendendo minimo il *rapporto di risposta* (ovvero il tempo impiegato ad eseguire il processo diviso per il tempo che si sarebbe impiegato se la macchina fosse stata completamente scarica).

Tra le **scelte di progetto** per gli algoritmi di allocazione dei processori (o meglio dei processi), si può optare innanzitutto tra *algoritmi migratori o non migratori*, a seconda che un processo in esecuzione su una CPU vi debba rimanere fino alla terminazione o possa essere spostato su un'altra CPU, *algoritmi deterministici o non deterministici* a seconda che si basino o meno su informazioni note a priori sui processi da eseguire, *algoritmi centralizzati o distribuiti* a seconda che la decisione venga presa da una sola macchina che elabora le informazioni in suo possesso oppure da un accordo fra più macchine, ed infine *algoritmi ottimi o sub-ottimi* a seconda se ci si accontenti di una soluzione accettabile oppure si cerchi sempre la migliore allocazione possibile, con grande costo per raccogliere più informazioni e processarle più a fondo.

Per i vari algoritmi bisogna dunque stabilire le **politiche di trasferimento**, ovvero quando un nuovo processo creato debba essere mandato in esecuzione sulla macchina su cui è stato generato e quando debba invece essere trasferito su di un'altra meno carica di lavoro, nonché le **politiche di posizionamento**, ovvero su quale macchina debba essere spedito un processo di cui si è deciso di sbarazzarsi.

Per realizzare effettivamente un algoritmo di allocazione dei processori, sorgono inoltre alcuni problemi pratici legati alle modalità con cui una macchina possa stabilire e misurare

quale sia il proprio attuale **carico di lavoro** e se debba ritenersi sovraccarica. In particolare, come abbiamo visto, contare il *numero di processi in esecuzione* non è un buon metro di valutazione in quanto potrebbero esservi molti demoni che in realtà aggiungono pochissimo carico al sistema. In alternativa può essere presa in considerazione la *frazione di tempo in cui la CPU è occupata*, interrompendola periodicamente per osservarne lo stato; purtroppo alcune parti critiche di codice disabilitano le interruzioni, ed in tal modo l'effettivo utilizzo della CPU sarà sempre sottostimato.

Un altro problema pratico deriva dalla necessità di stimare accuratamente l'**overhead introdotto dallo spostamento dei processi**, in quanto il costo della raccolta delle misure e dello spostamento effettivo potrebbero rimangiarsi tutto il guadagno ottenuto dall'avere spostato un processo su di una macchina meno carica di lavoro.

7 - Architettura del Laboratorio Virtuale

7.1 Il Laboratorio Virtuale come Sistema Distribuito

Per consentire agli utenti del Laboratorio Virtuale di ricevere vari tipi di servizi da macchine remote che gestiscono le varie risorse, il VL deve essere progettato come un vero e proprio **Sistema Distribuito**.

In particolare si desidera adoperare dei multicalcolatori, senza memoria condivisa ma connessi da una rete, su cui il laboratorio possa essere realizzato con del software ad accoppiamento stretto, realizzando nel contempo un **sistema virtualmente uniprocessore** per gli utenti.

Un tale sistema consentirebbe di godere di tutti i vantaggi dei Sistemi Distribuiti, tra i quali spiccherebbero, tra quelli già esaminati, la **distribuzione intrinseca** (alcune applicazioni come quelle nelle fabbriche sono intrinsecamente distribuite), la **condivisione di risorse** hardware e software (situate fisicamente in posizioni lontane dagli utenti ma rese ugualmente accessibili a tutti) e l'**efficienza** (per la possibilità di ripartire il carico di lavoro sulle macchine più appropriate).

Oltre alla già citata trasparenza che consentirebbe di celare agli utenti e magari anche ai programmatori il fatto che in realtà il Laboratorio Virtuale non è un sistema uniprocessore, un'altra importante caratteristica che dovrebbe essere inclusa nel progetto del VL dovrebbe essere la flessibilità, ovvero la possibilità di aggiornare con facilità i servizi offerti dal

laboratorio: questa caratteristica può essere ottenuta implementando le funzioni fondamentali in un **modello a Microkernel**, e tutti i vari servizi come processi server a livello utente.

Altre caratteristiche dei Sistemi Distribuiti da prendere in particolare considerazione per il progetto del Laboratorio Virtuale devono essere l'**affidabilità**, con la possibilità di variare il processore su cui le varie operazioni vengono eseguite al verificarsi di guasti, la **sicurezza**, con l'accertamento dell'identità di tutti gli utenti che richiedono servizi al VL, le **prestazioni**, scegliendo opportunamente la grana del parallelismo delle operazioni avviate su più processori, e la **scalabilità**, evitando dove possibile che il Laboratorio Virtuale si basi solo su componenti centralizzati.

7.2 Le comunicazioni nel sistema del Laboratorio Virtuale

Come già visto a proposito dei Sistemi Distribuiti in generale, la comunicazione tra due processi può essere basata su di un semplice protocollo, su di un modello Client/Server oppure sulle chiamate di procedura remota.

In realtà le comunicazioni basate solo su di un protocollo non hanno molta utilità e non specificano come strutturare il sistema; allora per il Laboratorio Virtuale ci si avvarrà quanto meno di un **modello Client/Server**, in cui i vari utenti si collegano col VL per chiedere la realizzazione remota di qualche servizio.

Realizzando il Laboratorio Virtuale come un modello Client/Server, allora, tutti gli scambi di informazioni tra i due processi coinvolti saranno realizzati con un semplice protocollo Request/Reply privo di connessione, con cui il client spedisce la propria richiesta di servizio ed il server risponde inviando il risultato della operazione.

Con tale modello tutte le operazioni consentite dal Microkernel si riconducono allora alle due primitive `send` e `receive`, ed il protocollo Request/Reply serve a stabilire le regole secondo cui devono essere rappresentate le richieste e le risposte.

Purtroppo, come già evidenziato, il modello Client/Server è particolarmente flessibile ma non gode per il programmatore della proprietà della trasparenza della distribuzione del sistema, in quanto chi programma il sistema dovrà essere perfettamente a conoscenza della distribuzione delle macchine sulla rete, dovrà spedire le proprie richieste indirizzandole al server ed attendere da esso le corrispondenti risposte, utilizzando le primitive `send` e `receive` per inviare e ricevere esplicitamente messaggi sulla rete.

Si potrebbe allora pensare di realizzare il Laboratorio Virtuale tramite delle **chiamate di procedura remota**, riguadagnando in tal modo il vantaggio della trasparenza e rendendo in tal modo il sistema virtualmente uniprocessore sia per gli utenti comuni che per i programmatori.

In tal caso il programmatore dovrà solo occuparsi del *parameter marshalling*, provvedendo alla corretta formattazione e rappresentazione dei parametri inseriti nella chiamata ed al corretto spacchettamento dei risultati al ritorno. La soluzione con l'utilizzo della RPC è quella che offre la maggiore astrazione in quanto anche per il programmatore diventano indistinguibili le chiamate locali da quelle remote.

7.3 Problemi dei modelli di comunicazione tramite RPC

Sia che per il VL si adotti un modello di comunicazione Client/Server, sia che si progetti mediante un insieme di RPC, sorgono alcuni problemi intrinsecamente legati al tipo di

servizi che come abbiamo visto nel capitolo 1 si desidera che il Laboratorio Virtuale offra agli utenti.

In particolare per realizzare dei servizi di ***remote monitoring*** oppure di ***remote performance evaluation*** su di impianti reali i modelli sopra esposti si presentano alquanto riduttivi per quanto riguarda il tipo di interazioni previste. Infatti, nel caso in cui si vogliano realizzare forme di monitoraggio o valutazione a distanza degli impianti, le interazioni tramite il protocollo Request/Reply o le RPC si dimostrano palesemente insufficienti in quanto inadatte a modellare il flusso di informazioni richiesto.

In queste situazioni, infatti, si richiede che in seguito ad una richiesta di un utente il server del VL cominci ad inviargli dati in tempo reale e continui così finché l'utente lo desideri. A tale necessità si adattano male sia il modello Client/Server, che con il protocollo Request/Reply senza connessione prevede l'invio di un singolo messaggio in risposta ad una richiesta di servizio, sia la RPC, in quanto una chiamata di procedura, sia pure remota, restituisce tutti in una volta i risultati della elaborazione.

A prima vista potrebbe sembrare che il VL possa essere facilmente ricondotto ad uno dei due modelli di comunicazione considerati semplicemente facendo in modo che sia lo stesso utente ad inviare ciclicamente una serie di *Request* al server, oppure ad invocare continuamente la stessa RPC; in realtà tale soluzione non è praticabile in quanto oltre ad essere poco elegante introduce sulla rete un notevole flusso superfluo di informazioni e costringe il server ad un continuo inutile lavoro di interpretazione dei messaggi che oltretutto potrebbe anche farne degradare il tempo di risposta.

Dovendo allora modificare uno dei due modelli di comunicazione possibili per il nostro laboratorio distribuito per incorporare le funzionalità richieste per il monitoraggio a distanza, è evidente come la RPC non possa essere in alcun modo espansa per il concetto stesso di chiamata di procedura, e che ci si debba concentrare sul modello Client/Server.

Mentre infatti la chiamata di procedura remota per sua stessa definizione è equivalente ad una analoga chiamata di procedura locale, e dunque solo al ritorno dalla elaborazione deve restituire tutti ed in una sola volta i risultati, non è invece detto che un modello Client/Server debba essere per forza incentrato esclusivamente su di un protocollo Request/Reply senza connessione, ma si possono prevedere al suo interno più protocolli selezionabili a seconda del tipo di servizio desiderato.

In particolare si rende necessario affiancare al protocollo Request/Reply un altro protocollo che renda possibile in maniera efficiente il rilevamento continuo del valore di alcune variabili di stato dell'impianto, senza che si renda necessariario che l'utente debba inviare periodicamente le richieste alla stessa frequenza con cui è richiesto il campionamento. Tale ulteriore protocollo sarà descritto nel prossimo paragrafo.

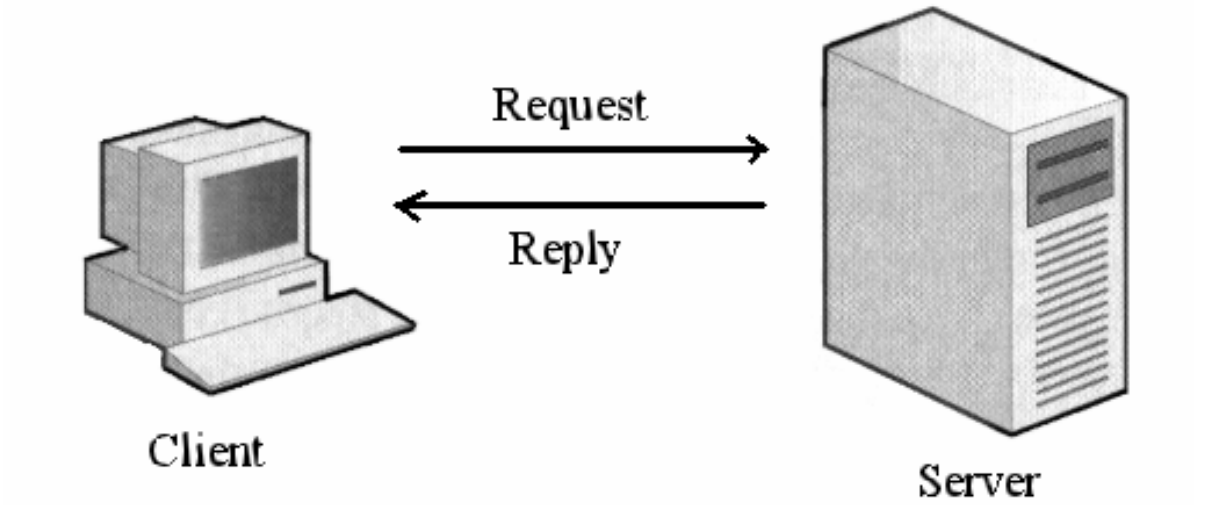
7.4 Protocolli di comunicazione del Laboratorio Virtuale

Si è visto come in definitiva, realizzando il Laboratorio Virtuale come un Sistema Distribuito che deve offrire certi servizi, bisogni basarsi su di un modello Client/Server, affiancando al protocollo Request/Reply un altro protocollo che renda possibile il monitoraggio in tempo reale dell'impianto.

In particolare tale nuovo protocollo dovrà fare in modo che ad una singola richiesta da parte del client il VL inizi a spedire periodicamente il valore delle variabili di cui è stato richiesto il monitoraggio, senza che l'utente effettui altre richieste e fino a quando una delle due parti non decida per qualche motivo di disconnettersi.

In base a tali richieste, bisognerà allora incorporare all'interno del Virtual Lab la possibilità di far comunicare tra loro il client ed il server secondo uno di due possibili protocolli:

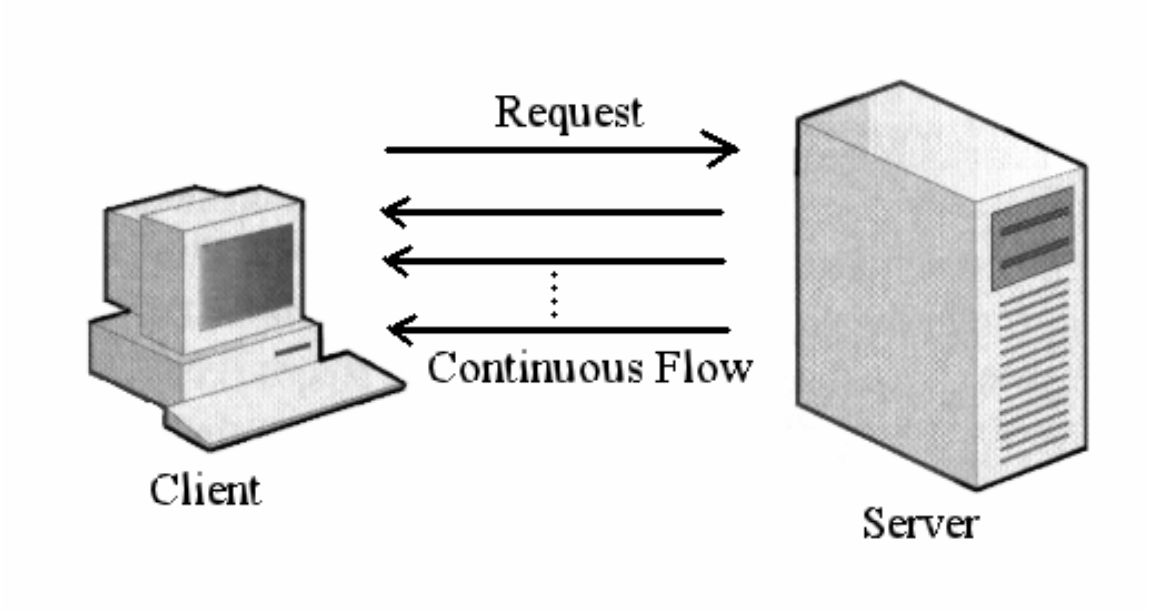
1. **protocollo Request/Reply**: ad ogni richiesta di servizio da parte del client corrisponde solo una risposta del server che restituisce in una sola volta il risultato dell'operazione. Questo protocollo si addice a quasi tutte le operazioni a distanza sugli impianti e agli accessi a simulatori e tutorial.



2. **protocollo Continuous Flow**: per permettere il monitoraggio in tempo reale delle variabili di interesse bisogna fare in modo che dopo una opportuna richiesta del client il server inizi a spedire periodicamente un flusso continuo di informazioni con i dati desiderati; dunque ad una sola richiesta da parte del client farà seguito una serie ciclica di risposte con i dati da monitorare aggiornati in tempo reale. Tale protocollo termina quando una delle due parti in comunicazione decide di disconnettersi.

Bisogna a tal proposito osservare come solitamente, nei modelli Client/Server esistenti, per il protocollo Request/Reply sia previsto l'utilizzo di un protocollo di comunicazione senza

connessione; nel nostro caso invece quantomeno per il protocollo Continuous Flow sarebbe auspicabile utilizzare un protocollo di comunicazione con connessione che provveda anche a far giungere all'utente i vari messaggi nell'ordine corretto.



La soluzione più generale, se il Laboratorio Virtuale vuole essere implementato con un modello a Microkernel, consiste nel prevedere in ogni caso l'instaurazione di una **connessione** tra il client ed il server, e la successiva spedizione dei messaggi desiderati all'interno di essa seguendo uno dei due possibili protocolli.

L'instaurazione di una connessione tornerà anche utile quando verranno analizzati i requisiti di sicurezza dell'impianto, al fine di facilitare l'autenticazione di tutte le richieste che giungono al client, e rende possibile fare in modo che alcune informazioni utili alla contrattazione preliminare che avviene tra le due parti non debba essere svolta per ogni singola richiesta ma solo all'atto dell'instaurazione di una nuova connessione, all'interno della quale il client possa effettuare più richieste.

7.5 Scelte di progetto

Una volta stabilito che il Laboratorio Virtuale è un Sistema Distribuito la cui architettura è organizzata secondo un modello Client/Server, e che dunque è possibile per gli utenti richiedere servizi che verranno eseguiti da un opportuno servente, che d'ora in poi chiameremo *Lab Server*, per definire completamente l'architettura è necessario compiere ulteriori scelte di progetto.

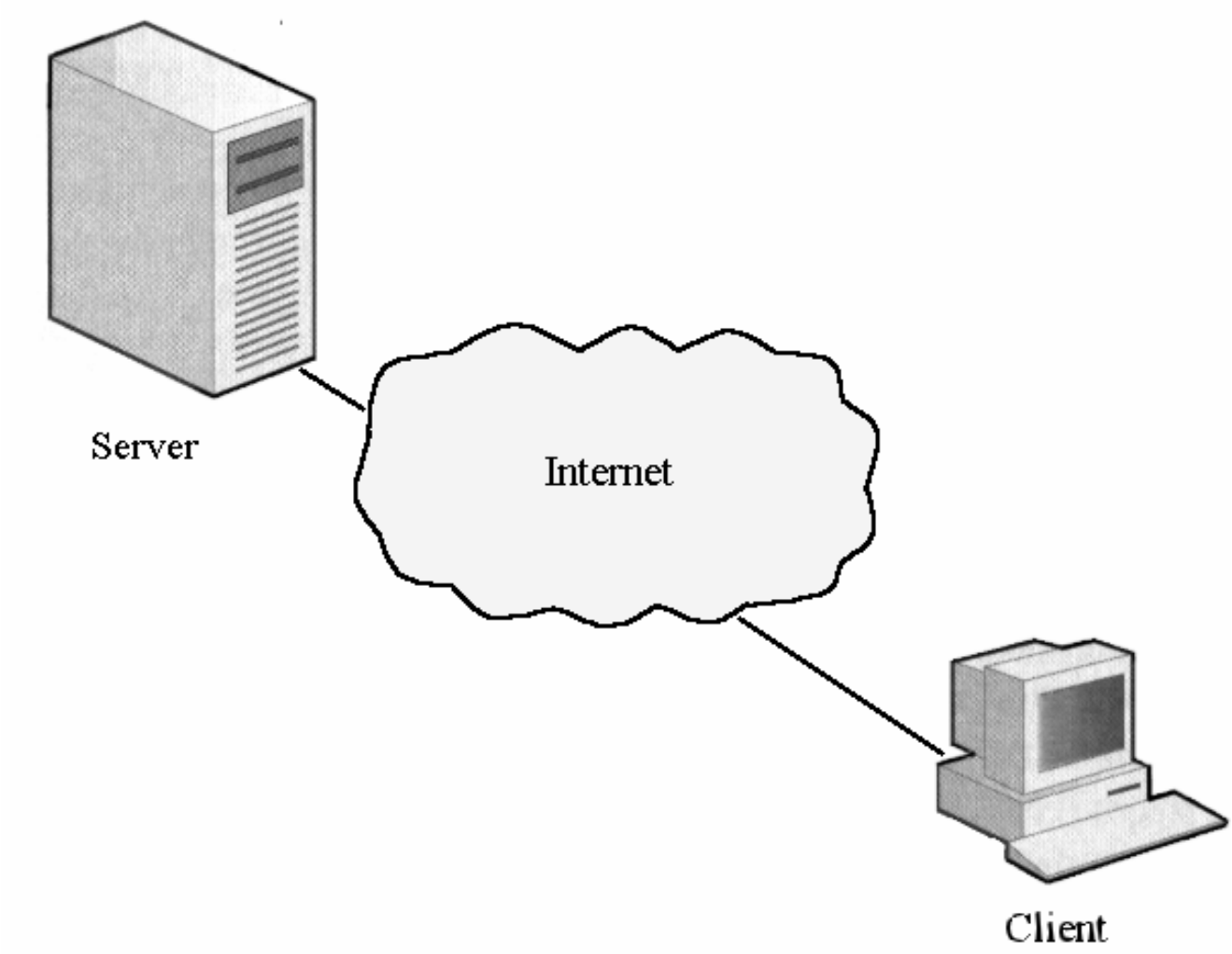
Un prima scelta di progetto riguarda le modalità di organizzazione del lavoro da parte del *Lab Server*, e sarà analizzato nel dettaglio nel seguente capitolo 8.

Altre scelte di progetto riguardano la definizione delle modalità di interazione tra l'utente e le varie risorse disponibili, in cui il Lab Server si trova a fare da tramite; la definizione delle interazioni possibili va sotto il nome di *interfaccia*. Le modalità di accesso alle risorse tramite le interfacce possibili sono presentate nel capitolo 9.

8 - Organizzazione del server

8.1 Scelte di progetto

Il Lab Server deve prendere in esame tutte le richieste di azione sulle singole risorse e deve agire di conseguenza. Su questo server possono essere compiute varie scelte di progetto, in particolare riguardo ai seguenti punti:



1. scelte di progetto riguardanti la eventuale **replicazione** del Lab Server;

2. scelte riguardanti le varie possibilità di **allocazione dei lavori** da parte del server su altre macchine;
3. scelte riguardanti i **compiti** del server all'interno del protocollo di comunicazione adottato.

In particolare riguardo alla **replicazione** si valuterà la possibilità che l'utente non abbia a disposizione un unico Lab Server cui sia possibile inviare le richieste, ma possa averne a disposizione più di uno a seconda delle proprie necessità; di questo aspetto si discuterà nel paragrafo seguente.

Invece come **allocazione dei lavori** (*job allocation*) si intende la possibilità che il Lab Server deleghi ad altre macchine l'esecuzione effettiva delle operazioni da svolgere, nei soli casi in cui ciò sia consentito e con le modalità più opportune; questo aspetto sarà approfondito a partire dal paragrafo 8.3.

Nel caso in cui i lavori siano allocati dal server su altre macchine, bisognerà precisare se il **compito** del server sia solo quello di ripartire inizialmente ogni richiesta sulle altre macchine, che provvedono poi dunque autonomamente a proseguire la comunicazione con l'utente, oppure se il server debba continuare a fare da tramite per tutta la durata della connessione; questo argomento sarà affrontato alla fine del capitolo.

8.2 Replicazione del server ed algoritmo di elezione

Riguardo al primo punto delle scelte di progetto riguardanti il Lab Server si fa notare come, organizzando il Sistema Distribuito con un unico server cui afferiscono tutte le richieste, l'architettura che in prospettiva si va delineando per il Laboratorio Virtuale sia fortemente

centralizzata e soffra di un inevitabile **collo di bottiglia** che può penalizzare le prestazioni per tutti gli utenti e portare il sistema in saturazione.

La soluzione che si potrebbe cercare di realizzare per superare il collo di bottiglia consiste nel **replicare** il Lab Server su più macchine host, cercando successivamente di fare in modo di distribuire nella maniera più equa possibile le richieste dei client tra i vari server, in modo tale da realizzare un sistema scalabile che permetta di aumentare incrementalmente il numero di server al crescere degli utenti collegati.

In generale, se esistono più server, ogni client deve disporre di un opportuno algoritmo da applicare per selezionare il server più appropriato cui inviare le proprie richieste. Tali algoritmi vengono solitamente detti **algoritmi di elezione** (o di *voting*).

Un obiettivo che un tale algoritmo potrebbe prefiggersi potrebbe essere quello di indicare ad ogni client quale tra i server disponibili sia al momento meno carico di lavoro, in modo da poter successivamente evadere le richieste del client il più rapidamente possibile.

Tale obiettivo viene di solito approssimativamente raggiunto applicando il seguente semplice algoritmo:

1. il client che vuole inoltrare una richiesta invia un messaggio in multicast a tutti i server chiedendo di inviare un semplice messaggio di risposta;
2. il client attenda la risposta di ognuno di essi;
3. appena arriva la prima risposta l'algoritmo termina;
4. il client legge l'indirizzo del server che ha inviato la risposta giunta per prima: questo server vince la contesa e sarà quello cui il client continuerà a comunicare inviando le proprie richieste;
5. i messaggi di risposta ricevuti dopo il primo vengono scartati.

Tale algoritmo basa generalmente la sua efficacia sull'assunto empirico che il primo server a rispondere sia presumibilmente quello più scarico di lavoro, ed è l'algoritmo usato ad esempio nel Network File System di Sun. In particolare, inoltre, applicato ad un Sistema Distribuito con hardware ad accoppiamento lasco quale è il Virtual Lab, in cui i vari server potrebbero essere distribuiti su scala mondiale, tale algoritmo si dimostra altrettanto efficace in quanto tiene conto, oltre che del carico di lavoro dei vari server, anche della distanza di essi dal client e della banda disponibile per la comunicazione.

In definitiva dunque l'efficacia di questo semplice algoritmo di elezione applicato dal client si basa sulla elevata probabilità che il Lab Server la cui risposta arrivi per prima sia quello che anche nel seguito possa garantire all'utente il minimo tempo di risposta.

8.3 Valutazione della possibilità di allocazione distribuita

Oltre alla organizzazione interna riguardante la eventuale replicazione del Lab Server su più host, bisogna valutare anche se tutte le richieste provenienti dai client debbano essere servite direttamente dal server oppure se questi potrebbe delegare l'esecuzione di alcuni lavori ad altre macchine messe appositamente a disposizione del Laboratorio Virtuale.

Inizieremo con l'osservare che per il Laboratorio Virtuale la scelta di adoperare un Sistema Distribuito, ed in particolare un modello di comunicazione Client/Server, delegando ad una delle due parti il compito di fungere da servente per compiere tutte le operazioni richieste, si rende indispensabile per i seguenti motivi a seconda delle risorse su cui ci si trova ad operare:

1. per quanto riguarda gli **impianti reali**, si è già osservato in generale a proposito dei Sistemi Distribuiti come molte applicazioni presentino la proprietà della *distribuzione intrinseca*; tra

tali applicazioni vanno ad esempio le transazioni bancarie, le prenotazioni di voli aerei e, ciò che ci riguarda, le applicazioni industriali; un'altra proprietà che può essere utile ottenere per gli impianti reali è la *condivisione*, in quanto un Sistema Distribuito può permettere a più utenti di accedere agli stessi impianti remotamente e talvolta, per la sola lettura delle variabili di stato, anche a più utenti contemporaneamente.

2. per quanto riguarda gli **strumenti di simulazione e valutazione automatica**, essi sono quasi sempre dipendenti dalla piattaforma su cui vengono ad essere eseguiti, e spesso molti tool necessitano di notevoli risorse di calcolo; può allora tornare comodo organizzare il Laboratorio Virtuale con un Sistema Distribuito Client/Server in modo da trarre vantaggio dalla proprietà della *efficienza*, ovvero delegando al server il compito di eseguire i vari simulatori sulle macchine più opportune; anche per questo secondo tipo di risorse vale il vantaggio della *condivisione*, che permette a tutti gli utenti di utilizzare allo stesso tempo tutti i tool software disponibili.

Allora, per quanto riguarda le macchine su cui il Lab Server si troverà a dover eseguire i vari comandi, osserviamo che mentre per gli **impianti industriali** la caratteristica della distribuzione intrinseca degli stessi non lascia adito a dubbi riguardo all'ubicazione delle macchine che si troveranno ad intervenire sulle variabili di stato, per quanto riguarda invece i **tool di simulazione** il Lab Server può avere un certo potere discrezionale, andando a distribuire il carico di lavoro su altre macchine cercando di ottimizzare qualche obiettivo.

In taluni casi l'allocazione delle elaborazioni su più macchine può essere una scelta obbligata, in virtù del fatto che molti simulatori e tool di valutazione possono essere eseguiti solo su determinate piattaforme; in tal caso se è disponibile solo una macchina con le caratteristiche necessarie il problema della scelta non si pone. Invece nel caso in cui sia possibile effettuare una scelta bisognerà studiare un apposito algoritmo di allocazione.

8.4 Algoritmi di allocazione dei lavori

Negli unici casi in cui il Lab Server abbia effettivamente un certo potere discrezionale per valutare quale delle macchine disponibili per l'esecuzione dei tool debba essere incaricata dell'esecuzione di ogni simulazione, bisognerà adottare un opportuno **algoritmo di allocazione dei lavori** (*job allocation*) sulle varie macchine a disposizione del Laboratorio Virtuale. Tale problema si presenta molto simile all'allocazione dei processi in un sistema multiprocessore, analizzata nel capitolo 6.

Prima di congegnare un apposito algoritmo, bisogna innanzitutto prefiggersi un obiettivo da ottimizzare; essendo il Laboratorio Virtuale un sistema distribuito interattivo, un buon **obiettivo** sarebbe *minimizzare il tempo di risposta* per i vari utenti, ovvero il tempo che intercorre tra la spedizione della richiesta dell'utente al proprio Lab Server e la ricezione della risposta con i risultati della elaborazione.

In tale caso, il tempo di risposta è dato dalla somma dei tempi dovuti alla trasmissione dei vari messaggi, alla schedulazione dei lavori ed al tempo di elaborazione totale, comprensivo anche dei tempi in cui il processo è pronto ma non è mandato in esecuzione; tra tutte queste componenti quella su cui è più facile intervenire è il tempo di processamento.

E' evidente come per minimizzare il tempo di processamento sarebbe conveniente che le simulazioni più pesanti dal punto di vista computazionale venissero assegnate alle macchine più potenti e più scariche di lavoro. Si è già osservato nel capitolo 6 come in realtà valutare l'effettivo carico di ogni macchina sia un problema non banale, ma è comunque possibile dare alcune direttive in base a cui elaborare un apposito algoritmo.

8.5 Algoritmo di job allocation

Come detto, l'algoritmo di job allocation dovrebbe cercare di assegnare le computazioni più pesanti alle macchine più potenti e più scariche di lavoro, ottenendo in tal modo per gli utenti il miglior tempo medio di risposta. Un buon algoritmo potrebbe essere una variante dell'*algoritmo ad asta* [TANE], adattato al caso in cui ogni processo da allocare ed ogni macchina disponibile abbiano delle caratteristiche note a priori e valutabili in maniera centralizzata.

Ad esempio, poiché le operazioni consentite sui vari tool di simulazione sono ben catalogate, si potrebbe far corrispondere ad ogni richiesta di servizio di un utente al Lab Server un opportuno coefficiente, tale da pesare opportunamente il costo dell'operazione dal punto di vista delle risorse di calcolo impegnate.

Tale coefficiente, che diremo **peso della operazione**, consentirebbe di poter stabilire una scala di valori su cui le varie richieste in arrivo al Lab Server potrebbero essere ordinate.

Il peso della operazione potrebbe essere ricavato dinamicamente sulla base del tipo di richiesta (che contribuisce staticamente) e del valore di alcuni parametri richiesti dall'utente (riguardanti ad esempio la durata e la complessità della simulazione):

$$W = W_s + W_d$$

Oltre alle operazioni, si potrebbero assegnare degli opportuni coefficienti anche alle varie macchine adatte per l'effettuazione dell'operazione richiesta, in modo da poter stabilire una scala di valori anche per le risorse di calcolo messe a disposizione del Laboratorio Virtuale.

Tale coefficiente, che diremo **capacità della macchina**, andrebbe anch'esso ricavato dinamicamente sulla base della potenza di calcolo del computer in questione (costante per ogni macchina) e del carico di lavoro sostenuto prima dell'allocazione:

$$C = C_s + C_d$$

Come misura del carico di lavoro, come visto nel capitolo 6, ci si potrebbe accontentare di ottenerne una misura dalla frazione di tempo per cui la CPU è occupata.

Dopo avere stabilito delle scale di valori per le operazioni richieste dagli utenti e per le macchine, l'algoritmo dovrebbe cercare di creare una sorta di **corrispondenza tra le due scale**, in modo dal allocare ogni lavoro su di una macchina la cui capacità non sia né troppo superiore né troppo inferiore a quella richiesta dal peso della operazione:

$$W \cong C$$

In definitiva l'algoritmo adottato dovrebbe essere non migratorio, deterministico, centralizzato e quasi ottimo. In particolare se il Lab Server riceve una richiesta di comando a distanza su di un impianto o di esecuzione di un tool di simulazione per il quale è disponibile una sola macchina, la scelta è obbligata e non c'è bisogno di applicare l'algoritmo; l'algoritmo di allocazione va dunque applicato solo nel caso in cui il Lab Server riceva una richiesta per l'esecuzione di un tool di simulazione che può essere avviato su più macchine disponibili, ed in tal caso dovrebbe procedere nel seguente modo:

1. la richiesta dell'utente deve essere interpretata, ed in particolare il peso della operazione dovrà essere ricavato dal peso statico associato al tipo di elaborazione, nonché dal valore di alcuni parametri impostati dall'utente;
2. il Lab Server dal tipo di operazione ricava anche informazioni riguardanti quali tra le macchine a sua disposizione per l'esecuzione sono adatte a volgere il lavoro richiesto;
3. il Lab Server provvede ad interrogare tutte queste macchine per ottenere in risposta il loro attuale carico di lavoro;

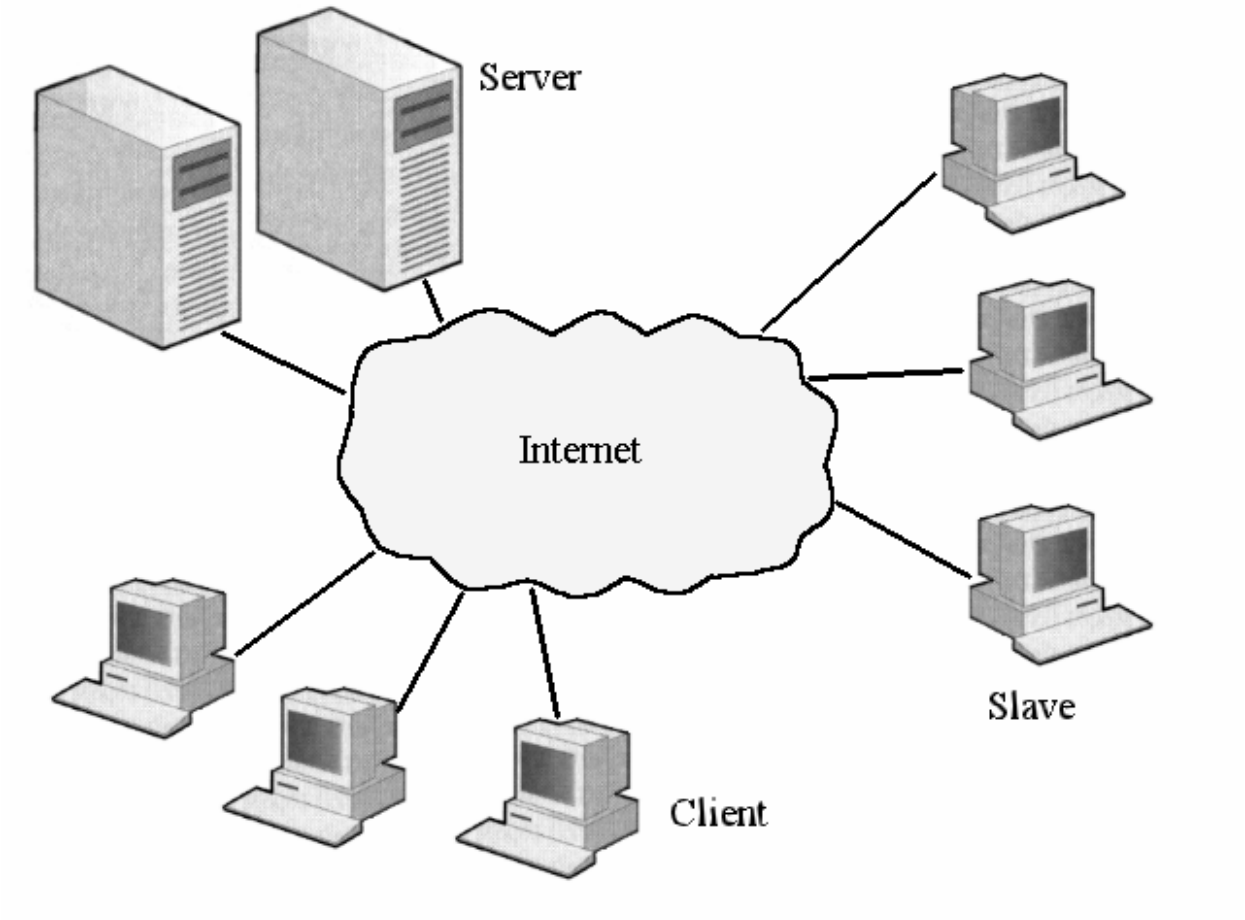
4. dalla potenza di calcolo associata ad ogni macchina e dall'attuale carico di lavoro, il Lab Server estrae la attuale capacità di calcolo delle macchine;
5. confrontando il peso della operazione con tutte le capacità delle macchine il Lab Server decide a quale tra le macchine allocare il lavoro richiesto dall'utente.

L'effettiva efficacia dell'applicazione di un simile algoritmo dovrebbe essere comunque valutata nell'applicazione ad un caso concreto; infatti molto dipende dai pesi che l'amministratore del sistema attribuisce arbitrariamente in fase di configurazione alle operazioni e alle macchine. Tali valori andrebbero dunque settati inizialmente e poi variati più volte, verificando empiricamente quali valori forniscano effettivamente le prestazioni migliori, fino ad ottenere una corretta calibrazione.

8.6 Organizzazione generale del Lab Server

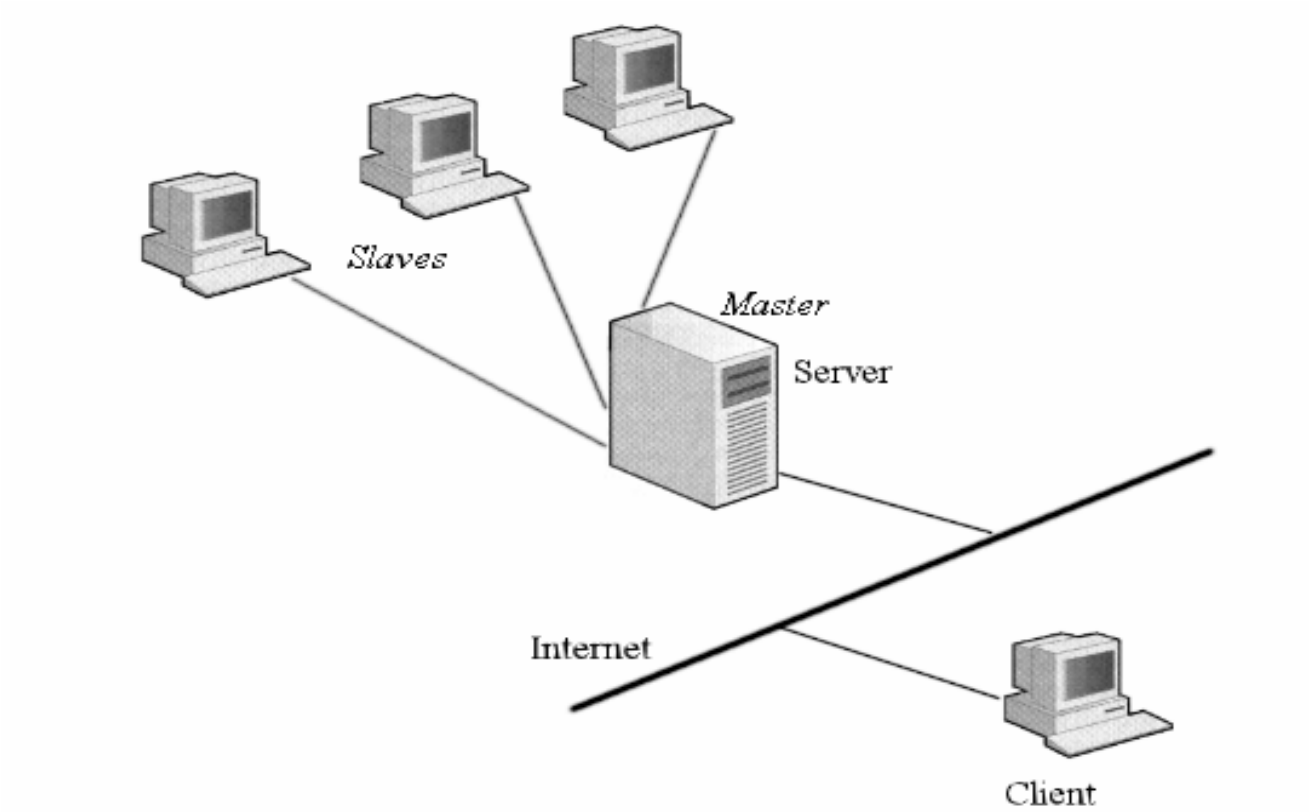
In definitiva la ripartizione dei lavori da parte del Lab Server avviene secondo una struttura *Master/Slave*, in cui il server (quello eletto tra tutti quelli possibili) provvede a distribuire il carico di lavoro tra vari computer slave che provvedono all'esecuzione dei vari comandi a distanza sugli impianti e delle varie simulazioni richieste dagli utenti.

In tale schema, il Lab Server può essere visto sia come un server rispetto ai vari client (gli utenti) che accedono dalla rete, sia come il master di una struttura di computer gerarchica che si occupa di eseguire tutte le operazioni richieste. Con tale schema il master di per sé non deve necessariamente avere la capacità di eseguire operazioni sulle risorse, ma provvede semplicemente a smistare le richieste di servizio alle varie stazioni di lavoro.



Come visto, le richieste di comandi a distanza vengono inviate direttamente allo slave presente nell'impianto di cui si vuole consentire l'accesso remoto, quando invece per le richieste di esecuzione remota di tool di simulazione il Lab Server provvede ad attuare l'algoritmo di allocazione per scegliere la macchina più appropriata cui delegare l'esecuzione del comando.

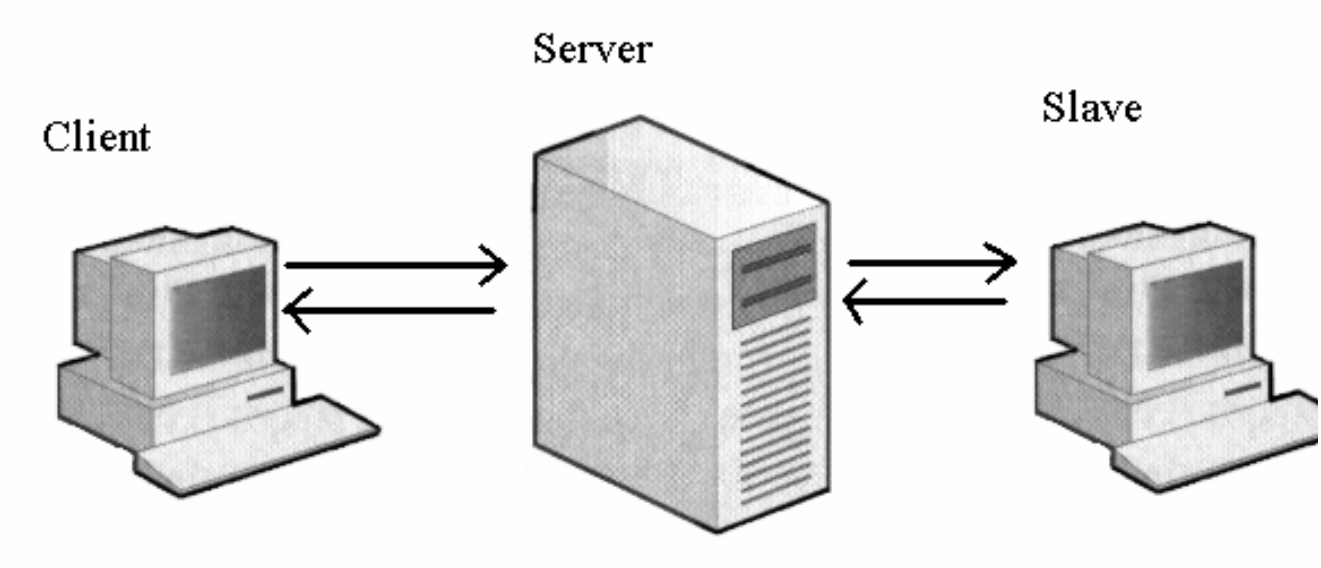
Nel caso più generale tutte le macchine considerate, ovvero i client, i server e gli slave, possono afferire alla stessa rete, ad esempio Internet. In certi casi può essere invece conveniente che gli slave non siano direttamente collegati ad Internet, ma possano mettersi in comunicazione solamente con gli slave tramite una Intranet: l'utilizzo di una rete privata garantisce, oltre che maggiori garanzie di sicurezza, anche comunicazioni molto più veloci con larghezze di banda elevate.



8.7 Compiti del server nel protocollo di comunicazione

Sulla base di quanto visto fino ad ora, il Lab Server verrebbe ad avere solamente delle funzioni di smistamento sui vari slave dei lavori richiesti dai client. Rimane allora da stabilire se, all'interno del protocollo di comunicazione, il Lab Server debba continuare a fare da tramite tra il client e lo slave prescelto per tutta la durata della comunicazione, oppure se una volta assegnato il lavoro possa farsi da parte e lasciare che la comunicazione prosegua direttamente tra il client e lo slave. Ognuna delle due possibilità presenta vantaggi e svantaggi che verranno nel seguito analizzati.

La prima possibilità, ovvero quella in cui il Lab Server **continua a fare da tramite** per tutta la durata della connessione, presenta i seguenti vantaggi:

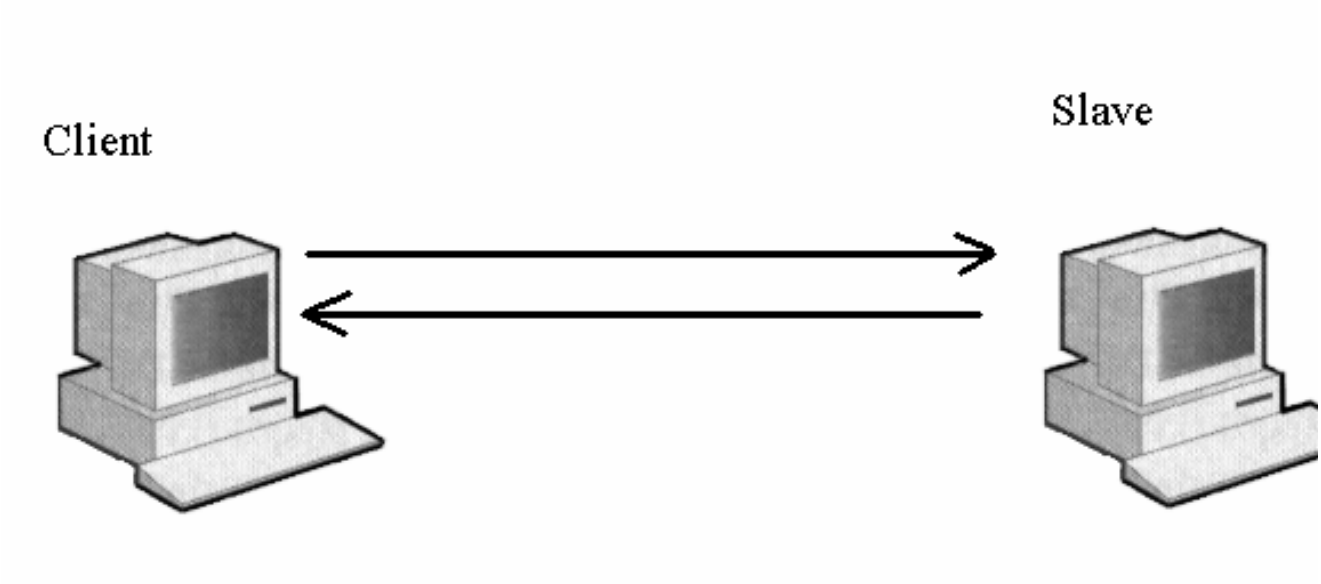


1. tutti i meccanismi di protezione, che verranno analizzati nel seguito, sono centralizzati all'interno del server, e sono più sicuri in quanto non si rende possibile ad un client di accedere direttamente ad uno slave;
2. in caso di esperimento congiunto dell'utente su più risorse, il Lab Server ha la visione generale del sistema e può verificare che le operazioni richieste sulle varie risorse siano tra loro compatibili;
3. il Lab Server conosce in ogni momento quanti lavori sono allocati su ogni macchina e può trarne informazioni da utilizzare per applicare l'algoritmo di job allocation.

D'altronde però tale modalità di comunicazione comporta i seguenti svantaggi:

1. tutte le richieste e tutte le risposte devono passare attraverso il Lab Server, che essendo centralizzato può appesantirsi fino ad andare in saturazione;
2. anche nel caso migliore il Lab Server introduce un certo ritardo.

Invece la seconda possibilità, ovvero quella di fare in modo che dopo una fase preliminare il Lab Server **si faccia da parte** e tutte le comunicazioni avvengano solamente tra il client e lo slave allocato, presenta i seguenti vantaggi:



1. la comunicazione risulta più veloce;
2. il Lab Server non ne risulta appesantito.

In compenso però sorgono ora i seguenti svantaggi:

1. il sistema è meno sicuro perché è possibile per un client accedere direttamente agli slave;
2. non vi è una entità centralizzata che abbia una conoscenza generale del sistema e possa controllare che le varie operazioni richieste contemporaneamente siano tra loro compatibili;
3. il Lab Server non dispone di alcuna informazione su quanto possa durare ogni singolo lavoro allocato, e dispone dunque di poche informazioni per riapplicare successivamente l'algoritmo di job allocation.

Delle due soluzioni, ognuna come detto ha i propri vantaggi e i propri svantaggi; in particolare però la prima massimizza la sicurezza a scapito delle prestazioni e della scalabilità, mentre la seconda punta sulle prestazioni a discapito di parte della sicurezza. Allora una possibile scelta andrà fatta, a seconda del particolare tipo di Laboratorio Virtuale da realizzare,

ricercando un compromesso in funzione dei requisiti di prestazioni e di sicurezza che si desidera ottenere.

Inoltre, a seconda delle scelte implementative adottate, la soluzione potrebbe essere obbligata; questo caso si verificherà ad esempio nella implementazione proposta nella terza sezione, in quanto per una GUI realizzata con una applet del linguaggio Java le uniche interazioni con l'esterno possono avvenire solamente tramite il server, e dunque per questa realizzazione particolare la scelta della prima possibilità di comunicazione si rivelerà indispensabile.

9 - Risorse ed interfacce

9.1 Le interfacce del laboratorio

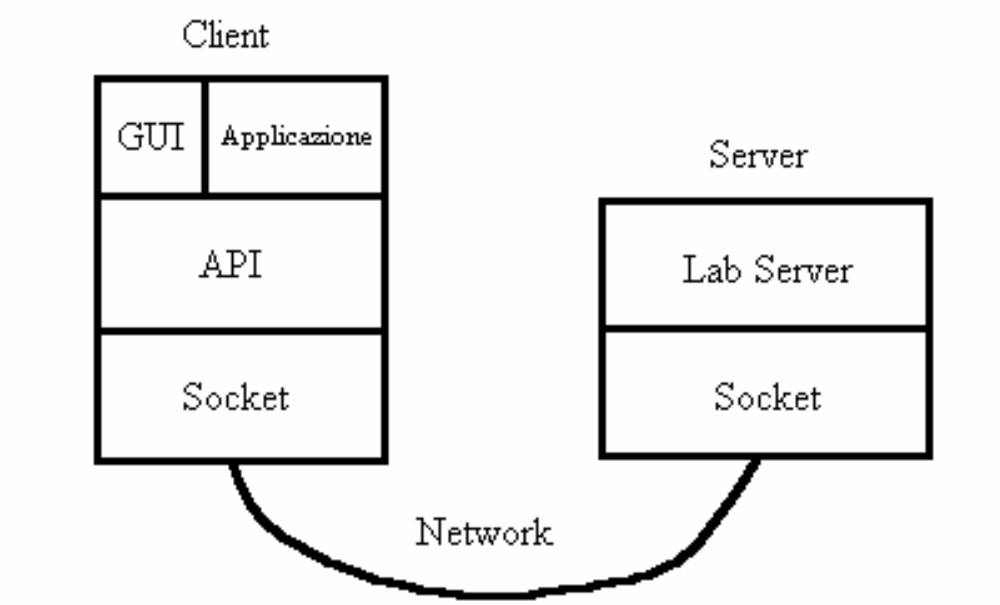
Come accennato nel capitolo 1, il Laboratorio Virtuale deve prevedere almeno due tipologie di interfacciamento da parte degli utenti.

Deve essere innanzitutto presente una interfaccia *user-friendly* con cui l'utente possa interagire in maniera immediata per compiere operazioni sulle risorse disponibili. Tali interfacce vengono solitamente realizzate in modalità grafica per permettere una grande velocità di accesso anche agli utenti più inesperti, e vengono denominate col nome di **GUI** (*Graphical User Interface*, interfaccia grafica per l'utente).

Il fatto che si tratti di una GUI non deve fare presupporre che essa possa essere utilizzata solo per scopi didattici o poco seri: anche (e soprattutto) per il tecnico specializzato di un impianto industriale è utile poter accedere al proprio impianto in maniera rapida tramite una interfaccia che presenti in maniera grafica i vari controlli per modificare le variabili di stato dell'impianto o per diagnosticare su quale componente possa essersi verificato un malfunzionamento.

Un altro tipo di interfaccia che il Laboratorio Virtuale deve presentare all'esterno è l'interfaccia per la programmazione delle applicazioni (**API**, *Application Programming Interface*). La API viene supportata per permettere agli sviluppatori di interagire con le risorse del laboratorio senza seguire lo schema precostituito della GUI, consentendo di creare dei

propri programmi che accedano anche a più risorse del laboratorio contemporaneamente, assemblando nella maniera più flessibile le interfacce che vengono offerte verso le varie risorse.



Viceversa rispetto alla GUI, in tal caso lo sviluppatore che voglia realizzare delle proprie applicazioni locali che interagiscano con le risorse del laboratorio deve avere una dettagliata conoscenza dell'interfaccia per la programmazione che gli viene offerta verso tali risorse.

C'è da osservare che in realtà la GUI non sarà una interfaccia indipendente dalla API, ma anzi può essere considerata la principale applicazione che ne faccia uso.

9.2 Accesso ai vari tipi di risorse

Per l'architettura distribuita del Laboratorio Virtuale che si va delineando bisogna ora analizzare le modalità di accesso alle varie risorse. Con questo si vuole indicare che le modalità di esecuzione remota di un tool di simulazione saranno certamente dissimili da quelle del monitoraggio di un impianto industriale.

Nel seguito saranno analizzate nell'ordine le interazioni che l'utente può effettuare sulle varie risorse, in particolare differenziando i casi a seconda del tipo di risorsa coinvolta e della interfaccia che l'utente utilizza:

1. interazione con un simulatore tramite la GUI;
2. interazione con un simulatore tramite la API;
3. accesso remoto ad un impianto tramite la GUI;
4. accesso remoto ad un impianto tramite la API.

Una nota particolare sarà dedicata al caso in cui l'utente desideri monitorare un impianto reale, ed in tal caso, come già visto nel capitolo precedente, si rende necessario l'utilizzo di un protocollo di tipo Continuous Flow; dove non altrimenti specificato si sottointenderà invece l'utilizzo di un comune protocollo Request/Reply.

9.3 Interazione con un simulatore

Nel caso in cui si voglia mettere a disposizione degli utenti del laboratorio un simulatore, bisogna stabilire quale tipo di simulatori possa essere reso disponibile e con quali modalità l'utente possa accedervi.

In generale, può essere incluso all'interno del Laboratorio Virtuale un qualunque simulatore che possa essere eseguito in modalità batch, ovvero che non richieda la presenza fisica di un utente interattivo al terminale della macchina su cui tale tool deve essere eseguito; in caso contrario è ovvio come non sia possibile nessuna forma di esecuzione remota.

Più in particolare, quasi tutti i simulatori permettono di essere eseguiti in modalità testuale semplicemente mediante passaggio dei parametri sulla riga di comando della shell; in

taluni casi invece dei parametri vi è la necessità di includere tutti i dati in ingresso all'interno di un opportuno file e di specificare il nome di tale file come parametro da passare al simulatore.

Alla fine della simulazione le modalità di visualizzazione dei risultati sono analoghe, in quanto tipicamente i dati in uscita vengono scritti su di un file oppure visualizzati sullo standard output, che può essere facilmente ridiretto verso la destinazione desiderata.

Tutti i tool che presentano queste tipologie di interazione possono essere facilmente inclusi all'interno del Laboratorio Virtuale, in quanto le interazioni del modello Client/Server si limiteranno al passaggio dei parametri da parte dell'utente remoto ed alla restituzione dei corrispondenti risultati.

In generale allora tutto ciò che il VL deve fornire sono l'interfaccia di ingresso, tramite cui l'utente può settare i parametri per richiedere l'esecuzione di una certa simulazione, e l'interfaccia di uscita, tramite cui vengono visualizzati i risultati della elaborazione.

9.4 Interazione con un simulatore tramite la GUI

Nel caso in cui l'utente desideri interagire tramite una GUI, l'interfaccia di ingresso e quella di uscita sono già state predisposte dal gestore del Laboratorio Virtuale; in tal caso l'utente deve limitarsi a settare in maniera elementare i parametri rappresentati graficamente nell'interfaccia di ingresso, in cui le varie scelte sono 'guidate' ed i vari componenti sono facilmente identificabili visualmente; un certo tempo dopo avere richiesto l'elaborazione, verrà presentata all'utente l'interfaccia di uscita, in cui l'output potrà essere visualizzato sotto forma di testo o di grafici di ogni tipo.

Una delle possibilità più interessanti offerte dalla interazione con dei tool di simulazione mediante la GUI potrebbe essere la creazione di 'package' di simulatori dello stesso genere che

presentino lo stesso tipo di I/O, ovvero accettino gli stessi parametri in ingresso e producano in uscita dei risultati codificati in maniera equivalente. In tal modo per l'utente è possibile effettuare più volte lo stesso tipo di simulazione con gli stessi parametri, variando solamente di volta in volta il 'motore' della simulazione, in modo da poter compiere una prova comparata dei risultati forniti.

9.5 Interazione con un simulatore tramite la API

Nel caso in cui qualche programmatore desideri sviluppare delle proprie applicazioni che si interfaccino direttamente con una o più risorse del laboratorio, l'interfacciamento avviene utilizzando la API fornita ad ogni utente del Laboratorio Virtuale ed opportunamente documentata.

Con l'interfacciamento tramite API, al programmatore vengono forniti, 'a scatola chiusa', dei file sorgenti (che ad esempio definiscono classi nel caso di programmazione ad oggetti), che egli potrà utilizzare all'interno delle proprie applicazioni (presumibilmente scritte nello stesso linguaggio di programmazione della API) richiamandone alcune procedure di libreria pubbliche per svolgere le funzioni fondamentali della comunicazione con il Lab Server.

In tal caso, la creazione e la gestione di eventuali interfacce di ingresso e di uscita è totalmente a carico del programmatore, in quanto con la API vengono fornite solamente le poche primitive da utilizzare per la comunicazione e la descrizione dettagliata del *parameter marshalling* richiesto per l'esecuzione di ogni differente comando.

9.6 Accesso remoto ad un impianto

Per valutare come sia possibile consentire agli utenti del Laboratorio Virtuale di interagire con un impianto reale, bisogna innanzitutto analizzare le caratteristiche che un tale impianto dovrebbe presentare e le varie modalità di accesso da parte degli utenti.

Senza particolareggiare il discorso effettuando a priori delle scelte riguardo alle modalità implementative, è comunque evidente come la possibilità di effettuare una connessione remota con un impianto sia subordinata dalla presenza su di un computer di necessari dispositivi, hardware e software, che gli consentano di interfacciarsi con dei dispositivi atti ad interagire con l'impianto reale in questione.

In particolare tale computer, che funge per l'impianto da 'ponte' per l'esterno, dovrà avere la capacità di interfacciarsi con sensori ed attuatori digitali montati sull'impianto. Un **sensore digitale** è un dispositivo che monitora l'impianto, misura le sue uscite e le converte in grandezze fisiche discrete che possono essere rilevate da un programma software; un **attuatore digitale** è invece un ulteriore dispositivo che permette, comandato opportunamente via software, di agire sull'impianto modificando i suoi riferimenti.

Oltre a questi elementari dispositivi che devono essere presenti su qualunque sistema elettronico utilizzato per il controllo digitale degli impianti industriali, tale computer dovrà anche essere connesso ad una rete in modo tale da consentire la comunicazione con il Laboratorio Virtuale per lo scambio dei dati richiesto.

Come già accennato nel precedente capitolo, mentre nel caso più elementare lo scambio di informazioni tra l'utente e tale computer (con il Lab Server a fare da tramite) può basarsi su di un semplice protocollo Request/Reply, con cui l'utente chiede l'esecuzione di una singola operazione ed ottiene in risposta tutti ed in una volta sola tutti i risultati desiderati, nel caso particolare in cui l'utente desideri effettuare una forma di *remote monitoring* per leggere in tempo reale l'andamento delle variabili di stato del sistema è necessario adottare un protocollo di tipo Continuous Flow.

Con tale protocollo, l'utente vuole continuare a ricevere i dati circa l'andamento delle grandezze di interesse fino a quando non chieda esplicitamente di sconnettersi o scada il tempo messogli a disposizione dal Laboratorio Virtuale.

Un altro problema che bisognerà analizzare sarà quello dei tempi di risposta che il VL può offrire, in quanto per molte applicazioni industriali, come ad esempio il *remote control*, il controllo dell'impianto va fatto on-line ed i tempi di ritardo introdotti dal VL dovrebbero garantire il non superamento di una certa soglia critica.

Per particolareggiare quanto detto fino ad ora, si osserverà come nello scenario tipico un computer situato nelle vicinanze dell'impianto potrebbe essere dotato di una porta di *interfaccia seriale* (come ad esempio la porta standard RS-232) la quale permette di comunicare con una vastissima gamma di dispositivi per il monitoraggio ed il controllo digitale.

Sulla stessa macchina dovrebbero poi essere presenti un comune programma eseguibile, scritto ad esempio in linguaggio C, che diremo ***Serial Port Manager***, usato localmente per accedere all'impianto tramite la suddetta porta (si rammenterà al proposito che il C dispone di una apposita libreria `serial.h`) e l'apposito software dello slave del Laboratorio Virtuale, che diremo ***Lab Stub***, sempre in esecuzione come processo demone, che localmente possa avviare il programma precedente e dalla rete possa comunicare col Lab Server (o direttamente con l'utente) per ricevere le richieste.

Nel caso si impieghi una simile configurazione, se l'utente utilizza un protocollo Request/Reply ad ogni richiesta in arrivo corrisponderà una sola esecuzione del Serial Port Manager da parte del Lab Stub con i parametri specificati nella richiesta, mentre se l'utente desidera operare mediante un protocollo Continuous Flow per monitorare l'impianto il Lab Stub dovrebbe eseguire ciclicamente il Serial Port Manager alla frequenza di campionamento richiesta oppure, per le applicazioni più tempo critiche, lo stesso Serial Port Manager dovrebbe

essere opportunamente programmato per leggere continuamente dalla porta seriale e passare al Lab Stub solo il compito di spedire i dati sulla rete.

9.7 Accesso remoto ad un impianto tramite la GUI

L’accesso remoto ad un impianto reale tramite la GUI è per l’utente virtualmente identico a quello ad un simulatore, in quanto le interfacce grafiche di ingresso e di uscita che vengono predisposte dal gestore del laboratorio possono essere progettate per svolgere le stesse funzioni.

In linea di massima, infatti, per le operazioni che l’utente richiede di eseguire in remoto e che possono basarsi sul protocollo Request/Reply, le uniche diversità sono dal lato del server, con il Lab Server che invece di avviare una simulazione richiede l’esecuzione di un comando su di un impianto, mentre il client, ovvero l’utente, desidera solo configurare opportunamente i riferimenti dell’impianto e leggere sull’interfaccia di uscita se l’operazione richiesta ha sortito l’effetto desiderato.

Invece, se l’utente sta compiendo una campagna di rilevamenti allo scopo di effettuare un monitoraggio remoto su alcune variabili dell’impianto, il flusso di informazioni coinvolto è continuo e non si basa sul protocollo Request/Reply, bensì sul protocollo Continuous Flow con cui, come abbiamo già visto, il Lab Server continuare con una certa frequenza di campionamento ad inviare dati al client; in tal caso l’interfaccia di uscita non dovrà limitarsi ad aspettare una sola volta il risultato della elaborazione e visualizzarlo, ma dovrà bloccarsi in un cliclo che le consenta di aggiornare la visualizzazione grafica dello stato delle variabili ad ogni pacchetto ricevuto dal server.

9.8 Accesso remoto ad un impianto tramite la API

L’utente può anche decidere di sviluppare delle proprie applicazioni per interagire con gli impianti presenti nel Laboratorio Virtuale senza utilizzare l’interfaccia grafica già predisposta, ma interfacciandosi direttamente con le procedure di libreria della API. In tal caso la creazione di eventuali interfacce grafiche di ingresso e di uscita è totalmente a carico del programmatore, in quanto le primitive della API permettono solamente di formattare opportunamente il messaggio da inviare al server.

Dopo che la richiesta opportunamente formulata è stato inviata al server, ciò che segue dipende dal tipo di protocollo richiesto dalla interazione.

Ciò che avviene nel caso del protocollo Request/Reply è abbastanza semplice, in quanto il programma dell’utente si limita ad inviare al Lab Server una richiesta opportunamente formattata e si blocca attendendo una risposta, che viene poi interpretata dall’interfaccia di uscita sviluppata *ad hoc* dal programmatore.

Ben più complicato è il caso in cui l’interazione richiesta sia del tipo *remote monitoring* e sia necessario utilizzare il protocollo di comunicazione Continuous Flow, in quanto in tal caso sarà l’utente che dovrà farsi carico di leggere ciclicamente i dati forniti dal Lab Server e di comunicare eventualmente a quest’ultimo la cessazione dell’esperimento.

Comunque, nel caso dell’accesso remoto ad un impianto reale tramite la API, i maggiori problemi non saranno dell’utente-programmatore, in quanto le procedure di libreria della API sono poche e ben documentate, bensì del Lab Server, in quanto dovrà continuamente verificare che una tale libertà di interazione lasciata agli utenti non possa generare un uso improprio e pericoloso dell’impianto.

A tal proposito, allora, bisognerà fare in modo che qualunque richiesta proveniente dai client debba essere accuratamente analizzata dal server, prima di essere eseguita, per verificare che non possa causare agli impianti danni derivanti da un uso improprio. Nel capitolo seguente si analizzeranno allora le varie problematiche di sicurezza che vengono a generarsi progettando una simile architettura distribuita: in particolare bisognerà partire dall'assunto che i client possano inviare richieste di qualunque tipo, anche palesemente pericolose, e che dunque tutti i controlli di sicurezza debbano in ogni caso essere sempre svolti dal server per qualunque richiesta in arrivo.

La creazione di appositi sbarramenti per le richieste provenienti dai client potrà dunque essere applicata non solo alle richieste provenienti dalle applicazioni autonome (basate sulle API), ma anche su quelle provenienti dalle GUI preconfigurate, non presupponendo in nessun caso a priori che le richieste provenienti dalle GUI possano godere di maggiori privilegi.

L'introduzione di vari controlli sulle richieste in arrivo potrà causare chiaramente dei ritardi nell'effettuazione dei comandi a distanza, ma tali controlli sono indispensabili e l'effettivo ritardo introdotto potrà essere valutato solo effettuando le opportune misure sulla particolare implementazione di Laboratorio Virtuale.

9.9 Riepilogo dello schema generale del sistema

Secondo quanto visto nel capitolo precedente, il Laboratorio Virtuale è un Sistema Distribuito organizzato secondo un modello Client/Server, in cui a sua volta il Lab Server provvede a ripartire il carico di lavoro in maniera distribuita tra vari slave.

A seconda del tipo di interazione richiesta dall'utente, allora, lo scambio di informazioni tra l'utente ed il Laboratorio Virtuale si svolge con la modalità seguente:

innanzitutto l'utente accede all'interfaccia di ingresso, la configura opportunamente ed invia la richiesta al server invocando la corretta procedura presente nella API; nel caso in cui l'accesso sia avvenuto tramite la GUI è essa stessa che provvede, in maniera del tutto trasparente per l'utente, ad invocare la primitiva della API.

Il server, una volta ricevuta la richiesta, la smista alla macchina più opportuna che provvede ad eseguire fisicamente il lavoro e ad inviargli il risultato; dopodiché il server invia il risultato della elaborazione al programma dell'utente, che si era bloccato attendendo la risposta e che provvede a visualizzarlo nella forma opportuna.

Come si vede, tutto si basa sulle semplici primitive `send` e `receive`, in quanto il client (sia esso una GUI od un programma dedicato) prima chiama una `send` contenente il servizio richiesto e poi si blocca su una `receive`, quando invece il server compie l'operazione inversa rimanendo in attesa di richieste su una `receive` e rispondendo eventualmente con una primitiva `send`.

Quanto detto vale ovviamente solo nel caso in cui si stia adoperando un protocollo Request/Reply, quando invece per il protocollo Continuous Flow sia il client che il server si bloccano in un ciclo in cui eseguono rispettivamente una `receive` ed una `send` fino a quando l'utente non faccia esplicita richiesta di disconnessione od i meccanismi di sicurezza del Laboratorio Virtuale non indichino che è l'utente ha esaurito il tempo a sua disposizione.

10 - Sicurezza del Laboratorio Virtuale

10.1 Requisiti di sicurezza del laboratorio

Una volta che si sia implementato un VL, una delle problematiche fondamentali è la sua sicurezza; con questo termine, nel caso del laboratorio, possiamo intendere tre cose:

1. **Outsider Access Security** (OAS, *sicurezza degli accessi dall'esterno*): si dovrà evitare che un utente della rete possa accedere alle risorse senza essere autorizzato ad accedere al laboratorio.
2. **Insider Access Security** (IAS, *sicurezza degli accessi dall'interno*): si dovrà fare in modo che ogni utente autorizzato per interagire con alcune risorse non possa accedere alle altre per le quali non è dotato dei necessari permessi.
3. **Intrinsic Lab Operation Security** (ILOS, *sicurezza intrinseca delle operazioni sul laboratorio*): bisognerà fare in modo che ogni utente, utilizzando una risorsa per la quale è autorizzato non possa in nessun modo, neanche per sbaglio, condurre il laboratorio a situazioni di pericolo.

Chiaramente la necessità di assicurare la sicurezza intrinseca vale solo per l'accesso alle risorse reali, quando invece gli strumenti di simulazione e valutazione automatica non possono in nessun modo venire danneggiati da un uso improprio; per questi strumenti software è invece necessaria la sicurezza dall'esterno e dall'interno per evitare l'accaparramento di costosi mezzi di calcolo.

Oltre a questi requisiti per la sicurezza del laboratorio, a volte lo stesso utente potrebbe desiderare che sia le proprie richieste di operazioni a distanza, e sia i relativi risultati, rimanessero segreti. Questo potrebbe ad esempio accadere nel caso in cui si preveda per l'utente una forma di pagamento per ogni operazione svolta dal laboratorio, nel qual caso l'utente desidererebbe chiaramente evitare che qualcun altro possa intercettare i risultati delle proprie elaborazioni senza avere pagato. Parleremo in questo caso, a proposito di questo ulteriore requisito di sicurezza richiesto al sistema, di *riservatezza dell'utente*.

10.2 La sicurezza dagli accessi esterni ed interni

Per garantire la sicurezza del laboratorio dall'esterno e dall'interno si desidera che ogni singolo utente che disponga dell'accesso a qualche risorsa del laboratorio venga catalogato in base ai propri diritti, e che possa in ogni caso interagire solo con le risorse del laboratorio cui egli ha il diritto di accedere. In questo contesto si è già definita come *risorsa* una qualunque parte del laboratorio con cui sia possibile interagire, sia essa un impianto reale od un mezzo di simulazione e valutazione automatica.

Invece qualunque altro utente non esplicitamente autorizzato deve ritenersi preclusa ogni forma di accesso.

Dunque i punti fondamentali in cui si articola la sicurezza del laboratorio sono i seguenti:

1. un utente non autorizzato non può accedere al laboratorio;
2. può accedere ad una risorsa del laboratorio solo un utente esplicitamente autorizzato per il suo utilizzo;

3. l’utente autorizzato ad accedere ad alcune risorse del laboratorio non può accedere alle altre risorse per le quali non è esplicitamente autorizzato.

Nel seguito saranno analizzate le principali direttive per garantire al laboratorio la sicurezza dall’esterno e dall’interno.

10.3 Meccanismi di protezione tramite autenticazione

Per fare accedere alle risorse del laboratorio solo gli utenti espressamente autorizzati, ovvero per realizzare la sicurezza sia external che internal, è necessario stabilire dei meccanismi di protezione tramite cui possa essere effettuata l’autenticazione degli utenti remoti.

Il metodo più comunemente adottato per l’autenticazione di ogni utente è quello basato sulla password, assegnando ad ognuno inizialmente una parola d’ordine riservata, della quale egli solo è a conoscenza, e la successiva corretta comunicazione della password al gestore del laboratorio implica necessariamente che l’utente sia veramente quello che dice di essere.

Il meccanismo di autenticazione del VL si basa su di un apposito protocollo che sarà analizzato nel dettaglio nel capitolo seguente. Esso consente di identificare con certezza gli utenti in modo da concedere ad ognuno di essi le autorizzazioni per interagire con le singole interfacce delle risorse del laboratorio. Si basa inoltre sugli algoritmi di cifratura per evitare che la password inviata dall’utente che tenta di accedere possa essere intercettata.

10.4 Politiche di differenziazione degli utenti

Una volta che l’utente sia stato autenticato egli consegue le autorizzazioni interagire con un sottoinsieme delle risorse del laboratorio.

Per ogni nuovo utente iscritto deve essere infatti creato un profilo differente, a cui corrisponda un particolare modello di accesso; l’applicazione di questo modello di accesso differenziato deve essere tale che l’utente che abbia accesso ad alcune risorse sia considerato esattamente un generico intruso nei confronti delle risorse di cui non possiede l’autorizzazione, ovvero la sicurezza dall’esterno può considerarsi un caso particolare di sicurezza dall’interno.

Per fare ciò è necessario che al momento della connessione ogni utente venga identificato ed autenticato, verificando poi quali autorizzazioni possiede sulle singole risorse del laboratorio, e tali permessi devono essere di volta in volta verificati per gli accessi ad ogni singola risorsa.

Nel caso che l’utente intenda accedere alla risorsa tramite la GUI, è possibile fare in modo che gli vengano presentate nell’interfaccia grafica solo le risorse del laboratorio su cui egli può vantare i necessari permessi di accesso, in modo da limitare al minimo la richiesta di accesso a risorse per le quali non si è autorizzati, anche se questo non dovrà in ogni caso far declinare l’obbligatorietà del controllo delle autorizzazioni su ogni richiesta di accesso.

Infatti, in tutti i casi, il Lab Server, che mantiene memoria di tutti gli utenti attualmente autenticati, ricevendo le richieste di comandi a distanza sulle varie risorse dovrà controllare ogni singola richiesta verificando che essa sia suffragata da un permesso prima di metterla in atto.

Questo è tanto più importante se si considera che le richieste potrebbero anche pervenire attraverso una API, che lascia allo sviluppatore grande libertà e permette di richiedere accessi anche alle risorse per le quali non si è autorizzati.

10.5 Granularità delle autorizzazioni

Una delle caratteristiche fondamentali che si richiede ad una tale catalogazione degli utenti con differenti profili di accesso alle varie risorse è la granularità del meccanismo di differenziazione: ovvero è utile che nella maschera degli accessi consentiti o meno ad ogni utente non siano inserite per intero le risorse del Laboratorio Virtuale (cioé non è conveniente che ogni utente possa accedere o a tutto un simulatore o un impianto o un tutorial oppure non possa accedervi per nulla), ma è preferibile che sia possibile concedere ad ogni utente l'accesso a parti od operazioni sulle risorse negandogli magari l'accesso ad altre parti della stessa risorsa.

Un esempio di profili di accesso differenziati per i vari utenti potrebbe essere quello in cui gruppi di studenti possono accedere selettivamente alla lettura dei tutorial e all'utilizzo dei simulatori inerenti le materie di cui stanno seguendo i corsi, i docenti possono accedere ai tutorial anche col permesso di scrittura, e docenti e ricercatori possono accedere agli impianti reali messi a disposizione dalle imprese.

Una volta che l'utente viene autenticato correttamente, bisogna fare in modo che possa accedere solo alle risorse cui ha diritto e solo nei modi consentiti; questo problema è tipicamente risolto con un opportuno Modello di Sicurezza, la cui applicazione al VL sarà analizzata in dettaglio nel capitolo 12.

Una ulteriore caratteristica richiesta è che il modello di sicurezza implementi delle strutture che consentano di stabilire ad esempio quando il sistema sia sovrautilizzato, in modo da poter rifiutare nuovi accessi, sia per motivi di efficienza che per evitare eventuali attacchi del tipo *"denial of service"* da parte di un utente della rete che cerchi di accaparrarsi tutte le risorse del laboratorio per renderle indisponibili agli altri. In tal modo, ponendo un tetto limite al numero di utenti che possono collegarsi contemporaneamente, il laboratorio potrà garantire agli utenti che sono già collegati una elevata qualità del servizio, soprattutto in termini di velocità di esecuzione dei simulatori.

D’altronde gli utenti già collegati non possono utilizzare per troppo tempo le risorse, poiché ne priverebbero altri utenti in attesa, per cui il laboratorio deve implementare un meccanismo di ‘espulsione’ per quegli utenti che sono collegati da troppo tempo nel caso in cui si sia già dovuta rifiutare qualche nuova connessione per sovraffollamento.

Tutte le problematiche sopra elencate troveranno soluzione all’interno del Modello di Sicurezza che sarà descritto approfonditamente nel capitolo 12.

10.6 La sicurezza intrinseca degli impianti

Il laboratorio non deve solo fornire l’accesso a tutti gli utenti autorizzati che chiedano di instaurare una connessione, ma deve anche garantire il rispetto della sicurezza intrinseca delle operazioni sul laboratorio.

Questo problema come detto si presenta solo per l’interazione dell’utente con le risorse reali; per mettere in atto la sicurezza intrinseca è opportuno che il Lab Server, prima di compiere ogni singola operazione, verifichi che essa non possa generare situazioni di pericolo.

Si rende allora necessario all’interno del Lab Server un modulo preposto alla verifica delle condizioni di sicurezza per ogni singola operazione da compiere; tale modulo non dovrà limitarsi ad analizzare la correttezza formale delle richieste in arrivo ma dovrà prendere delle iniziative proprie andando ad esempio a verificare lo stato attuale di alcune variabili degli impianti prima di stabilire se la richiesta di comando remoto può essere perniciosa o meno.

Si dovrà allora implementare all’interno del Laboratorio Virtuale un modulo, che diremo *Security Manager*, che di volta in volta verifichi se l’operazione da intraprendere sia fidata o meno, sulla base di quanto sarà affrontato nel prossimo paragrafo; tale modulo dovrà inoltre implementare il Modello di Sicurezza che sarà descritto nel capitolo 12.

10.7 Meccanismi di controllo del Security Manager

In particolare i meccanismi di controllo incorporati nel Security Manager del laboratorio dovranno verificare che per ogni comando di cui sia stata richiesta l'esecuzione vengano soddisfatte le seguenti proprietà:

1. **Correttezza statica**: ogni valore da imporre all'impianto su cui agire non deve valicare i limiti fisici imposti dal costruttore; questa proprietà è facilmente verificabile ma è sufficiente solo nel caso in cui tali limiti siano noti a priori e soprattutto siano indipendenti dalle attuali condizioni delle altre variabili di stato dell'impianto.
2. **Correttezza dinamica**: se i valori che possono essere imposti ai riferimenti dell'impianto non sono ognuno indipendente dagli altri, per verificare che tale proprietà sia soddisfatta bisogna leggere lo stato attuale delle variabili di stato, calcolare in tempo reale il range di valori consentiti come nuovi riferimenti e verificare se la richiesta fatta dall'utente remoto cade all'interno dell'intervallo utile.

Il Security Manager realizzato non dovrà dunque effettuare semplicemente una computazione passiva per accertare il verificarsi di certe condizioni matematiche, ma dovrà poter autonomamente intervenire sugli impianti per effettuare delle campagne autonome di rilevazioni.

In tale modo si rende sicura anche la realizzazione di esperimenti congiunti su più risorse, in quanto le correlazioni presenti tra più impianti o tra le varie parti dello stesso impianto possono essere modellate all'interno del Security Manager.

Realizzando tale Security Manager come una entità programmabile in un linguaggio di alto livello sarà allora possibile discriminare nel miglior modo possibile quali operazioni siano con certezza sicure e quali no, includendo magari al suo interno del codice per confrontare i parametri delle varie esecuzioni remote in corso o realizzando un monitoraggio per uso interno andando a leggere appositamente il valore di qualche variabile di stato.

In tal modo il programma del Security Manager scritto da chi ha reso disponibili le risorse, dovrà stabilire se l'effettuazione congiunta di più esperimenti possa rivelarsi dannosa per gli impianti coinvolti.

In ogni caso però la lettura delle variabili di stato e la successiva eventuale imposizione delle specifiche devono essere compiute come una singola azione atomica, in modo tale che non sia possibile intervenire su di un impianto mentre il Security Manager ha compiuto una rilevazione autonoma per valutare la correttezza di una operazione e debba ancora eseguirla.

Il fatto che le decisioni del Security Manager debbano essere implementate come azioni atomiche comporta le seguenti conseguenze:

1. i tempi di risposta del Security Manager devono essere molto stringenti;
2. gli accessi alle risorse devono poter essere bloccati dal Security Manager per il lasso di tempo che intercorre tra la lettura delle attuali variabili di stato e l'eventuale imposizione delle nuove specifiche.

Per quanto riguarda il primo punto, si può osservare che esso deve essere implementato in maniera tanto più stringente quanto più lo richiede la dinamica dell'impianto coinvolto, ad esempio si corre il rischio che nel momento in cui si vanno ad imporre alcuni riferimenti il valore delle variabili di stato non sia più quello che si era precedentemente rilevato.

Allora è probabile che un approccio simile possa essere realizzato per permettere il controllo remoto di un impianto chimico o petrolchimico, che notoriamente dispone di una

dinamica molto lenta, quando invece potrebbe essere particolarmente critico o addirittura irrealizzabile per impianti con dinamiche dello stesso ordine di grandezza di quelle richieste per la comunicazione col VL e per la successiva elaborazione dei dati acquisiti.

Per quanto invece riguarda il secondo punto, il bloccaggio delle risorse è una possibilità che deve essere resa necessaria anche per permettere la condivisione delle stesse risorse da parte di più utenti, che sarà presa in esame nel paragrafo seguente.

In ogni caso la semantica del meccanismo di bloccaggio nel caso in cui un utente trovi bloccata una delle risorse cui chiede di accedere dovrebbe fare fallire la richiesta piuttosto che porre l'utente in attesa che la risorsa occupata si liberi, in maniera tale, come si vedrà più approfonditamente nel capitolo 12 a proposito del Modello di Sicurezza, da evitare problemi di stallo.

10.8 Condivisione delle risorse

Un ulteriore problema riguarda la condivisione delle stesse risorse da parte di più utenti; infatti anche se le variabili di stato degli impianti possono essere lette da più utenti contemporaneamente, la loro modifica deve essere opportunamente regolarizzata, in modo che una loro eventuale concorrenza non comporti il danneggiamento dei rispettivi esperimenti o dell'intero impianto.

Dovranno essere allora previsti dei meccanismi di bloccaggio (*locking*) dei metodi utilizzati per accedere alle strutture dati degli impianti (reali o simulati). Tali meccanismi di locking saranno semplici da implementare, ma scomodi per l'utente, se la granularità del meccanismo è grossa, ovvero se l'utilizzo di una risorsa da parte di un utente preclude l'accesso a tutti gli altri utenti in attesa che ne facciano richiesta. Viceversa si desidera

implementare dei meccanismi con una granularità abbastanza fine, che consenta a quanti più utenti possibile di utilizzare parti differenti della stessa risorsa senza venire in conflitto.

A tale scopo il Lab Server dovrà tenere nota di quali utenti stiano in ogni momento accedendo agli impianti; nella maggior parte dei casi, se gli accessi sono del tipo domanda-risposta, il bloccaggio durerà relativamente poco, giusto il tempo di imporre le nuove specifiche e di rilevare il nuovo stato delle variabili; se invece la forma di accesso richiesta è il monitoraggio dell'impianto bisognerà distinguere due ulteriori casi:

1. se l'utente richiede solo di monitorare con continuità l'impianto, senza imporre su di esso nuove specifiche (con una sorta di modalità *read-only*) non vi è necessità di bloccare l'accesso a tale risorsa;
2. se invece l'utente oltre a monitorare lo stato delle variabili dell'impianto vuole anche di tanto in tanto interagire con esso imponendo nuovi riferimenti (modalità *read-write*) l'impianto dovrà essere bloccato e così dovrà rimanere per tutta la durata dell'esperimento dell'utente.

E' chiaro come si renda necessaria qualche forma di limitazione al tempo per cui ogni risorsa può rimanere bloccata da parte di ogni utente; questo problema verrà analizzato nel capitolo 12 in cui si presenteranno le modalità di differenziazione tra i vari utenti compiute dal modello di sicurezza implementato all'interno del VL.

Prima però sarà necessario analizzare le modalità con cui il Lab Server possa effettivamente essere sicuro dell'identità degli utenti che effettuano la connessione, ovvero come venga realizzata la sicurezza del laboratorio dall'esterno e dall'interno; tali requisiti di sicurezza vanno perseguiti non solo per l'interfacciamento con impianti reali, ma anche per l'accesso agli strumenti di simulazione e valutazione automatica, in modo da impedire che risorse di calcolo non indifferenti possano essere occupate da utenti non autorizzati.

10.9 Conclusioni

Questo capitolo ha posto le basi per tutta una serie di direttive di progettazione che saranno approfondite con maggiore dettaglio nei prossimi capitoli, ed in particolare nel capitolo 12 in cui sarà progettato un Modello di Sicurezza applicabile ad un generico Laboratorio Virtuale.

Il capitolo seguente inizia invece ad approfondire le problematiche di sicurezza a partire dal protocollo di autenticazione che permette al Lab Server di accertarsi dell'identità dell'utente posto dall'altra parte della connessione; tale capitolo va analizzato prima del seguente in quanto il Modello di Sicurezza si baserà su di esso, in quanto tutti i meccanismi di sicurezza che il modello implementa partono dall'assunto che l'identità del client possa essere identificata con certezza.

11 - Protocollo di autenticazione

11.1 Generalità

Il problema da analizzare è quello in cui il gestore del laboratorio vuole che ogni utente, collegandosi in rete con l'host remoto, possa dimostrare con certezza la propria identità, per consentirgli di interagire con le interfacce delle risorse su cui ha il diritto di eseguire le varie operazioni.

A seconda delle modalità con cui viene effettuata l'autenticazioni degli utenti, questo protocollo può costituire il principale mezzo per perseguire la Outsider Access Security, ovvero la proprietà secondo cui un utente non registrato presso il VL non può accedere a nessuna risorsa.

Questo non vuol necessariamente dire che tale proprietà sia condizione sufficiente per poter affermare che venga soddisfatta anche la Insider Access Security, in quanto bisognerà anche fare in modo che ogni utente regolarmente registrato non possa fare in modo di accedere al VL sotto l'identità di altri utenti registrati.

In ogni caso, mentre la Outsider Access Security si esaurisce con l'implementazione del protocollo di autenticazione, la Internal richiede anche che un opportuno modello di sicurezza, che sarà analizzato nel capitolo seguente, provveda a tener nota delle autorizzazioni da conferire ad ogni utente correttamente autenticato.

Dunque in prima analisi bisogna valutare le modalità con cui un utente viene registrato tra coloro che possono accedere ad alcune risorse del laboratorio, ed in seguito si vedrà come

si possa implementare ad ogni accesso uno scambio di informazioni tale da consentire la corretta autenticazione dell'utente.

11.2 Fase di iscrizione dell'utente

La fase preliminare del protocollo è quella in cui avviene la registrazione dell'utente. Al momento della iscrizione formale al laboratorio, il gestore del VL (che chiameremo *Lab Master*) fornisce all'utente che ne faccia richiesta (magari a pagamento) una login ed una password riservata, che sono le uniche informazioni di cui l'utente dovrà disporre per poter accedere al laboratorio tramite la GUI o da una propria applicazione (tramite la API).

Dunque l'iscrizione consiste nell'accesso da parte del Lab Master al laboratorio, ed alla sua richiesta di accredito di un nuovo utente; a questo proposito si può optare tra diverse scelte di progetto:

1. innanzitutto si può decidere se il Lab Master debba essere a conoscenza della password in chiaro;
2. bisogna vedere inoltre se il processo del Lab Server debba memorizzare esso stesso tale password in chiaro o meno (come ad esempio Unix che come detto registra nel file `/etc/passwd` solo le password cifrate);
3. si deve inoltre stabilire se il Lab Master debba compiere tale operazione necessariamente da un nodo prestabilito della rete (ad esempio direttamente sull'host su cui è eseguito il processo server del laboratorio) oppure possa compiere tale operazione da qualunque computer;
4. si deve infine decidere se l'utente stesso da iscrivere debba essere presente al momento dell'iscrizione insieme al Lab Master o se possa essere collegato da un qualunque nodo

remoto (anche questo influisce sulle modalità di trasmissione, cifratura e memorizzazione della password inserita dall'utente).

Bisogna osservare che per tutte e quattro le scelte esiste una soluzione realizzabile, ma chiaramente le scelte che lasciano maggiore libertà d'azione presentano una sicurezza inferiore, e dunque si potrebbero correre alcuni rischi.

Al momento dell'iscrizione vengono fissate le risorse del laboratorio a cui l'utente può accedere, ed in particolare l'utente potrà avere o meno accesso alle varie operazioni da svolgere sulle diverse risorse.

11.3 Utilizzo dei due algoritmi

Per implementare tutti i più moderni protocolli di autenticazione sono necessari entrambi gli algoritmi, quello simmetrico e quello asimmetrico a chiave pubblica.

A regime viene impiegato l'algoritmo simmetrico, che è quello che fornisce le maggiori garanzie sulla non decifrabilità dei messaggi anche nel caso in cui un intruso riesca ad intercettare e collezionare una grande quantità di messaggi.

Purtroppo, dal momento che per l'algoritmo simmetrico tutta la sicurezza risiede nella chiave e le due parti per comunicare devono concordare sulla stessa chiave da utilizzare, è necessario un metodo con cui preliminarmente tale chiave possa essere scambiata senza essere inviata in chiaro sulla rete.

Come detto, a tale scopo vengono comunemente impiegati gli algoritmi asimmetrici a chiave pubblica, con i quali il server può comunicare la propria chiave pubblica, in modo che

ogni informazione che venga cifrata dai client possa essere decifrata solo dal server stesso che dispone anche della chiave privata.

Tipicamente infatti, nei generici protocolli in cui due entità A e B vogliono comunicare, i due algoritmi vengono utilizzati nel modo seguente:

1. A chiede a B la sua chiave pubblica e la riceve in risposta in chiaro;
2. A sceglie una chiave di sessione casuale, la cifra con la chiave pubblica di B usando quel dato algoritmo asimmetrico e la spedisce a B;
3. B decifra il messaggio usando la sua chiave privata;
4. A e B possono iniziare a conversare utilizzando un algoritmo simmetrico e la chiave di sessione che ora conoscono entrambi.

11.4 Il generatore di chiavi casuali

Per completare il protocollo, oltre ad applicare i due algoritmi standard di cifratura simmetrica e asimmetrica, è necessario anche un generatore di chiavi casuali, da utilizzare come chiavi di sessione per l'algoritmo simmetrico. Cioé il server ad ogni richiesta di connessione genererà una chiave casuale per comunicare con ogni singolo client, la invierà al client con l'algoritmo a chiave pubblica (es. RSA) e utilizzerà questa chiave casuale per continuare a dialogare con il client mediante l'algoritmo simmetrico (es. DES).

Tale chiave casuale viene indicata con il termine di *chiave di sessione* in quanto avrà senso solo per la durata della singola connessione di ogni client con il Lab Server, dopodiché al momento della disconnessione essa verrà distrutta ed ogni eventuale messaggio in arrivo utilizzante quella chiave verrà scartato. Tipicamente il tempo di vita di ogni chiave di sessione sarà dato dalla durata della singola connessione da parte di un utente, ovvero da zero fino ad

alcuni minuti, con un limite superiore costituito dal massimo tempo di collegamento concesso ad ogni utente dai meccanismi di monitoraggio dello sfruttamento delle risorse del laboratorio.

Per generare casualmente una chiave di sessione è possibile utilizzare il generatore pseudo-casuale presente in tutti i linguaggi di programmazione di alto livello, in cui il periodo di ricorrenza è sufficientemente elevato. Si utilizza il termine *pseudo-casuale* perché quasi tutti i linguaggi che forniscono un generatore random calcolano in realtà i numeri in base ad algoritmi predeterminati, per cui periodicamente (dopo un certo periodo di ricorrenza) le sequenze di chiavi generate si ripetono.

Fortunatamente altri linguaggi, invece, come ad esempio Java, dispongono di un generatore di numeri casuali in cui il numero casuale generato non dipende solo dal numero precedente, e dunque prima o poi le sequenze possono ripetersi, ma il generatore viene reinizializzato ad ogni estrazione sulla base dell'orario attuale, calcolato in millisecondi, ed è in tal modo virtualmente impossibile riuscire a prevedere quale numero potrà essere estratto anche basandosi sulle osservazioni precedenti.

Tali chiavi casuali, come detto, sono chiavi di sessione, e come tali vengono memorizzate nella apposita struttura per ogni accesso corrente e vengono cancellate quando termina la connessione; in tal modo sarà utilizzata per la comunicazione una chiave differente non solo per ognuno degli utenti ma anche per ogni singolo accesso dello stesso utente.

11.5 Prosieguo della comunicazione mediante ticket

Come analizzato nel capitolo inerente l'architettura del Laboratorio Virtuale, la modalità di comunicazione nel modello Client/Server considerato prevede che l'interazione tra

l’utente ed il Lab Server per richiedere l’effettuazione di una operazione remota possa avvenire tramite un semplice protocollo Request/Reply o tramite un protocollo Continuous Flow.

Ogni utente, allora, all’atto della connessione con il laboratorio dovrà essere autenticato con il protocollo appena esposto, basato sui due algoritmi di cifratura dei dati simmetrico ed a chiave pubblica, e successivamente desidererà richiedere l’effettuazione di un comando a distanza mediante una `Request`.

Affinché il protocollo di autenticazione sia efficace, è necessario prevedere almeno due modalità attraverso cui l’utente possa garantire il VL che la propria richiesta proviene effettivamente da un utente autenticato:

1. la Request dell’utente deve essere cifrata utilizzando la stessa chiave simmetrica di sessione utilizzata nel protocollo di cifratura (ed anche la risposta o le relative risposte inviate all’utente saranno cifrate utilizzando la medesima chiave);
2. affinché il Lab Server possa subito rendersi conto se la richiesta è veritiera, una volta decifratala con la chiave prevista deve avere un modo immediato per rendersi conto se si tratta di una richiesta fidata o meno, andando ad identificare nella richiesta un campo che contenga una informazione per la quale si aspetti un certo valore.

Ad esempio insieme ad ogni `Request` potrebbero essere cifrate ogni volta le informazioni che l’utente digita per il proprio login, ovvero il suo User Name e la sua Password; però nella pratica risulta molto più comodo adottare un meccanismo di autenticazione basato su **ticket** (*riscontro*) per ogni singola richiesta, che agisca nel modo seguente:

1. all’atto della autenticazione, il Lab Server, oltre a generare casualmente una chiave di sessione da adoperare per tutta la durata della connessione, genera anche un numero casuale, detto *ticket*, e ne prende nota;

2. tale ticket viene inviato all'utente all'interno del protocollo di autenticazione congiuntamente alla chiave di sessione;
3. l'utente aggiunge ad ogni `Request` anche un campo contenente il ticket, e poi cifra tutta la richiesta utilizzando la chiave di sessione.

In tal modo il Lab Server, ricevuta ogni richiesta, avrà la possibilità di decifrarla e di stabilire immediatamente se essa provenga effettivamente dall'utente autenticato. Sono inoltre possibili variazioni per accrescere ulteriormente l'efficacia del protocollo complessivo, ad esempio facendo in modo che il server invii cifrato in ogni risposta anche un nuovo ticket che sostituisca il precedente, facendo sì che il tempo di vita di ogni ticket sia limitato ad una sola richiesta.

11.6 Schema del protocollo

Nel seguito, assumendo che come algoritmi di cifratura simmetrica e a chiave pubblica vengano adottati rispettivamente il DES e RSA, si adotterà la convenzione seguente:

- '`C`' è il client (cioé l'utente);
- '`S`' è il server (cioé il Lab Server);
- '`(n,e)`' è la chiave pubblica dell'algoritmo RSA;
- '`(n,d)`' è la chiave privata dell'algoritmo RSA;
- 'RSA_e' è il processo di cifratura dell'algoritmo a chiave pubblica RSA;
- 'RSA_d' è il processo di decifrazione dell'algoritmo a chiave pubblica RSA;
- '`session`' indica la chiave di sessione da utilizzare per l'algoritmo simmetrico DES;
- 'DES_{key}' indica il processo di cifratura simmetrico con chiave 'key';

- ‘ticket’ è l’identificatore che il server fornisce al client per farsi riconoscere;
- ‘pw’ sono il nome di login e la password dell’utente;
- ‘accesses’ è la comunicazione dei permessi conferiti all’utente.

Con la convenzione precedente, il protocollo di autenticazione dell’utente da parte del Lab Server dovrebbe in definitiva agire come segue:

1. il client spedisce in chiaro al server una generica richiesta di connessione:

   C --- “connect” ---> S

2. il server inizializza l’algoritmo RSA generando la coppia di chiavi pubblica/privata ed invia in chiaro la chiave pubblica:

   C <--- (n,e) --- S

3. il client genera casualmente (coi metodi della API) una chiave di sessione da utilizzare per il successivo algoritmo simmetrico DES, la cifra con la chiave pubblica dell’algoritmo RSA e la spedisce al server:

   $C \text{ --- } RSA_e(session) \text{ ---> } S$

4. il server decifra il messaggio usando la propria chiave privata RSA e ricava la chiave di sessione inviatagli dal client:

   $RSA_d(RSA_e(session)) = session$

5. il server genera casualmente un identificatore che il client dovrà utilizzare nelle successive comunicazioni e glielo invia cifrato col DES e la chiave di sessione:

   $C \text{ <--- } DES_{session}(ticket) \text{ --- } S$

6. il client decifra il messaggio ricevuto mediante l’algoritmo simmetrico e ricava l’identificatore assegnatogli:

   $DES_{session}(DES_{session}(ticket)) = ticket$

7. il client spedisce al server il ticket, il nome e la password dell'utente cifrandola con l'algoritmo simmetrico e la chiave di sessione:

$$C \text{ --- } DES_{session}(ticket,pw) \text{ ---> } S$$

8. il server decifra la richiesta utilizzando la solita chiave di sessione:

$$DES_{session}(DES_{session}(ticket,pw)) = (ticket,pw)$$

9. il server verifica che ticket, login e password siano corretti, controlla i permessi dell'utente e gli comunica il risultato del login:

$$C \text{ <--- } DES_{session}(accesses) \text{ --- } S$$

10. il client decifra il messaggio mediante l'algoritmo simmetrico e ricava i permessi che gli sono stati conferiti:

$$DES_{session}(DES_{session}(accesses)) = accesses$$

11. la comunicazione prosegue con il client che invia le proprie richieste al server, mediante l'algoritmo simmetrico e la chiave di sessione, accompagnandole con il proprio identificativo:

$$C \text{ --- } DES_{session}(ticket,request) \text{ ---> } S$$

12. ad ogni richiesta del client, il server risponde utilizzando lo stesso algoritmo simmetrico e la stessa chiave di sessione:

$$C \text{ <--- } DES_{session}(reply) \text{ --- } S$$

Client e server possono così continuare a comunicare finché l'utente continua ad interagire con l'interfaccia, e quando il client chiude la connessione oppure i meccanismi di sicurezza decidono di espellere l'utente (ad esempio per superamento dei limiti di tempo concessi) la connessione viene chiusa e la chiave di sessione non viene più considerata valida.

C'è da notare come quasi tutta la complessità dell'algoritmo possa essere nascosta non solo all'utente della GUI, ma anche per lo sviluppatore che utilizza la API, all'interno della API

stessa, in quanto l'unica operazione che deve effettivamente essere delegata all'utente è la digitazione di user name e password, nonché la pura affettuazione delle richieste dei comandi a distanza desiderati.

Il protocollo di autenticazione presentato si basa in parte su altri comuni protocolli che utilizzano la doppia chiave [SCHN], ma l'introduzione di altri meccanismi di riscontro, come i ticket, è stata appositamente introdotta per il Laboratorio Virtuale per permettere al Lab Server di autenticare una per una anche le successive richieste di esecuzione remota inviate dai client.

12 - Modello di sicurezza per il laboratorio

12.1 Il modello per il Laboratorio Virtuale

Il modello di sicurezza da adoperare per il Laboratorio Virtuale è un opportuno riadattamento del modello BLP analizzato nel capitolo 5.

Le strutture dati che si ritengono necessarie sono le seguenti:

- **Access Matrix ‘`M`’**: analogamente al modello BLP contiene in corrispondenza della coppia utente/risorsa le relative autorizzazioni;
- **Current Access Set ‘`a`’**: analogamente al modello BLP contiene le coppie utente/risorsa che descrivono gli accessi attualmente in atto, nonché altre informazioni utili per il monitoraggio degli accessi correnti;
- **User History ‘`u`’**: è una nuova struttura dati che si rende necessaria per memorizzare le password degli utenti e che può anche essere utilizzata per contenere informazioni riguardanti il monitoraggio degli accessi globali da parte di ogni utente;
- **Current Blocking Set ‘`b`’**: questa nuova struttura serve a memorizzare in ogni istante eventuali bloccaggi delle risorse oggetto di qualche accesso;
- **Resources Interaction Set ‘`i`’**: contiene le informazioni utili a definire le interazioni tra le varie risorse ed i limiti di funzionamento degli impianti.

Tali strutture dati saranno meglio analizzate nel seguito; non si ritengono invece necessari metodi per associare dinamicamente i livelli di sicurezza agli utenti e alle risorse,

intendendo per proibito qualunque accesso non esplicitamente indicato nella matrice degli accessi.

In tal modo, mantenendo tutti gli utenti e tutte le risorse allo stesso livello di priorità (a parte ovviamente il Lab Master che può compiere molte più operazioni degli altri utenti e può agire su tutte le risorse) le strutture dati risultano notevolmente semplificate rispetto al modello BLP di partenza; inoltre non si perde nulla dei requisiti di sicurezza introdotti in quanto con la sola Access Matrix e la opportuna granularizzazione dei permessi possono essere realizzate politiche di accesso molto restrittive.

Come evidenziato in [DBSE], ogni Modello di Sicurezza deve essere opportunamente adattato alla particolare applicazione sulla base delle proprie esigenze. A parte l'Access Matrix ed il Current Access Set, le altre strutture dati sono state progettate appositamente per l'utilizzo all'interno del Laboratorio Virtuale (per i motivi che verranno man mano presentati), e dunque di esse non si trovano riferimenti in letteratura.

12.2 Access Matrix

La struttura dati fondamentale del modello di sicurezza del Laboratorio Virtuale è costituita dalla *Access Matrix* '`M`' (matrice degli accessi), che memorizza in corrispondenza delle coppie `(s,o)` come il soggetto '`s`' può agire sull'oggetto '`o`'.

Nel nostro caso i soggetti sono i singoli utenti che possono accedere al laboratorio, mentre gli oggetti saranno i metodi mediante cui accedere a 'parti' di una risorsa del laboratorio. Infatti mentre '`s`' sono gli utenti regolarmente registrati presso il VL, saranno considerati oggetti '`o`' le singole risorse presenti nel laboratorio oppure, utilizzando una grana più fine, tutte le operazioni consentite sulle varie risorse.

Ad esempio, una operazione concessa su di un tool di simulazione può essere l'esecuzione, mentre per un impianto si avranno operazioni per monitorare le variabili di stato e per agire sugli attuatori, mentre per i tutorial le operazioni saranno quelli per la sola lettura o anche per la scrittura.

In corrispondenza di una coppia soggetto/oggetto, l'elemento della matrice `M(s,o)` sarà nel caso più elementare costituito da elementi booleani, che indicheranno se l'utente ha il diritto per accedere all'operazione concessa su di una determinata risorsa o meno.

Allora ad ogni utente verrà consentito l'accesso solo ad un *subset* dei possibili metodi, con una 'maschera' che viene stabilita dal gestore del laboratorio al momento della iscrizione e relativa registrazione dell'utente.

Ad esempio, ad uno studente potrà essere consentito solo l'accesso all'operazione relativa alla lettura di un tutorial, mentre ad un docente potrà essere consentito anche l'accesso per l'operazione di scrittura.

Con questa implementazione si ottiene una Access Matrix in cui sulle colonne sono disposte le singole operazioni per accedere alle risorse del laboratorio, e con questo si ottiene una grana particolarmente fine (la più fine possibile) per l'accesso alle varie risorse del laboratorio.

Questa matrice non viene modificata durante il naturale utilizzo del VL, ma solamente quando il Lab Master procede alla iscrizione di nuovi utenti, configurando opportunamente la maschera dei permessi, oppure quando egli stesso procede alla modifica dei permessi già conferiti precedentemente a qualche utente già registrato.

12.3 Propagazione delle autorizzazioni

E’ possibile pensare a schemi più complessi, in cui gli elementi della Access Matrix non siano puramente booleani; questo è utile soprattutto per progettare un meccanismo di propagazione delle autorizzazioni.

Piuttosto che prevedere solo una gestione centralizzata delle iscrizioni al laboratorio sarebbe utile pensare a dei meccanismi di propagazione; ad esempio il gestore potrebbe voler cedere ad altre strutture remote (ad esempio Università estere) l’onere di assegnare login, password e la ‘maschera’ degli accessi consentiti ad ogni singolo utente di quell’ente remoto.

In questo caso, la matrice degli accessi da implementare non sarà più booleana, ma dovrà prevedere per ogni metodo anche un identificatore che dica se l’utente in questione ha il permesso di propagare l’accesso a quel metodo.

In questo caso il generico elemento `M(s,o)` della matrice potrà assumere tre possibili valori:

- **Permesso consentito e propagabile:** è la modalità con cui accede il gestore del laboratorio ed ogni eventuale altro ente che abbia il diritto di iscrivere al laboratorio nuovi utenti.
- **Permesso consentito:** è il diritto che viene concesso ad un generico utente per farlo accedere ad alcune risorse del laboratorio.
- **Permesso negato:** l’utente ‘`s`’ non può utilizzare il metodo ‘`o`’ per accedere al laboratorio.

La possibilità di inserire nella Access Matrix anche i permessi propagabili è stata introdotta per consentire di delegare ad altri la possibilità di effettuare le iscrizioni per i nuovi utenti del Laboratorio Virtuale, e non è comunemente presente negli altri modelli di sicurezza che utilizzano la matrice degli accessi.

Nel caso di autorizzazioni propagate le iscrizioni non devono essere per forza effettuate dall’unico Lab Master, e dunque le operazioni relative alle nuove iscrizioni devono poter essere

compiute da qualunque nodo della rete Internet; allora le procedure di registrazione degli utenti dovranno esse stesse essere rese sicure e poter essere eseguite in remoto.

12.4 Current Access Set

Parallelamente alla matrice degli accessi è necessario sviluppare un'altra struttura, ovvero il *Current Access Set* '`a`', che tenga conto di quali utenti '`s`' stanno accedendo in ogni momento alle varie risorse '`o`' del laboratorio. Dunque tale struttura viene aggiornata man mano che gli utenti acquisiscono e rilasciano le varie risorse.

Questa struttura dovrà contenere, oltre all'insieme degli accessi attualmente in atto da parte degli utenti, anche le chiavi di sessione correntemente utilizzate per ogni connessione, nonché l'orario in cui la connessione ha avuto inizio.

Poiché tale struttura viene continuamente aggiornata e può essere azzerata in caso di caduta (*crash*) del sistema, potrà essere mantenuta dal Lab Server in memoria, anziché su disco, per cui non si presenta il problema di dover cifrare le chiavi di sessione, che andranno semplicemente memorizzate in apposite variabili private del server.

Tale struttura dati può essere utilizzata per attuare dei controlli sull'accesso al laboratorio: ad esempio si vorrà probabilmente limitare il numero massimo degli utenti che possono accedere contemporaneamente al laboratorio, e parallelamente si vorrà limitare anche la quota di risorse di sistema che il singolo utente può tenere impegnate, il tutto per non fare degradare troppo le prestazioni lasciando che un solo utente monopolizzi tutte le risorse di calcolo (per i simulatori).

12.5 User History

Oltre alla Access Matrix e al Current Access Set è necessaria una ulteriore struttura dati che contenga le login e le corrispondenti password di tutti gli utenti che possono accedere al laboratorio.

In questa struttura ‘u’, detta *User History*, possono anche essere memorizzate informazioni riguardanti le modalità passate di accesso del singolo utente, come ad esempio il tempo totale per cui l’utente è stato collegato al laboratorio, in modo da favorire per il laboratorio l’eventuale implementazione di un meccanismo di accounting che consenta forme di pagamento di entità proporzionale al tempo di effettivo utilizzo.

Così come il Current Access Set, questa struttura viene continuamente modificata man mano che l’utente accede alle varie risorse.

12.6 Current Blocking Set

Il *Current Blocking Set*, ‘b’, è una struttura che serve a memorizzare in ogni istante quali siano le risorse attualmente bloccate da una operazione in corso.

Le risorse bloccate possono ovviamente essere solamente gli impianti reali, in quanto di un tool di simulazione possono essere avviate più copie contemporaneamente. Come visto nel capitolo 10 a proposito della sicurezza intrinseca e della condivisione delle stesse risorse da parte di più utenti, il modulo che si occupa di bloccare eventualmente le varie risorse è il Lab Security Manager, che provvede dunque a registrare in ‘b’ l’inizio di un bloccaggio di una risorsa e provvede a cancellarne il corrispondente elemento qualora la risorsa venga rilasciata.

Il bloccaggio delle risorse non è chiaramente automatico ma va effettuato solamente per gli accessi alle risorse reali che richiedano una lettura-scrittura delle variabili di stato.

Tale struttura è sempre vuota tranne per pochi istanti durante le azioni atomiche compiute dal server; solamente nel caso in cui l'utente richieda di effettuare una forma di *monitoring* su qualche risorsa, mediante un protocollo Continuous Flow, vi saranno degli elementi presenti stabilmente. Chiaramente nel caso di monitoraggi continui sarà utile prevedere dopo un certo tempo massimo di utilizzo (verificandolo nel Current Access Set) una forma di rilascio obbligato della risorsa, per permettere anche ad altri utenti di poter interagire con essa.

Da notare che, secondo quanto detto nel capitolo 10, il bloccaggio di una risorsa non impedisce che altri utenti possano contemporaneamente accedere ad essa in modalità di sola lettura.

Nel caso in cui un client tenti di accedere ad una risorsa bloccata, si potrebbe fare fallire la sua richiesta oppure metterlo in attesa; considerato il tipo di accessi interattivi al Laboratorio Virtuale, la soluzione migliore consiste nel far fallire la chiamata inviando nel contempo all'utente l'indicazione del motivo del fallimento, in modo che l'utente possa riprovare a rieffettuare la chiamata in seguito.

In tal modo si eliminano anche le situazioni di stallo, in quanto un utente che voglia effettuare un esperimento congiunto su più risorse dovrà dichiararle tutte insieme al momento della richiesta, e nel caso in cui ne trovi anche una sola bloccata la sua richiesta fallirà completamente e non riceverà accesso a nessuna di esse.

12.7 Resources Interaction Set

Il *Resources Interaction Set* '`i`' costituisce l'insieme delle regole e delle leggi tramite cui le varie parti di un impianto oppure due impianti interagiscono influenzandosi

reciprocamente. Questo vuol dire che tale struttura non dovrà essere una struttura passiva, aggiornata man mano dal Lab Server, ma dovrà essere una struttura attiva che codifichi all'interno di una classe come variano le proprietà di ogni risorsa a seconda del valore delle variabili di stato proprie e di altre risorse.

Questo insieme di regole dovrà essere codificato all'interno di un modulo programmato per permettere al Lab Security Manager di accedervi e di stabilire se una richiesta di accesso possa essere considerata sicura o meno. Il Lab Server, prima di ogni operazione, interpella dunque il Lab Security Manager, il quale a sua volta consulta le regole all'interno di '`i`' per stabilire se l'operazione da compiere sia lecita o meno.

La memorizzazione dell'insieme di regole che sanciscono le correlazioni tra le diverse variabili dell'impianto può in realtà essere compiuto in due modi:

1. scrivendo tutte le regole all'interno di un file di testo con un formalismo da definire; in tal modo si ha il vantaggio che l'aggiornamento può essere compiuto in automatico, però la formalizzazione delle regole e la successiva ritrasformazione in condizioni da verificare all'interno del codice è tutt'altro che banale;
2. come detto, la soluzione da preferire risulta quella di inglobare tutte le regole all'interno di un programma cui il Security Manager si può poi riferire invocando il metodo opportuno; tale metodo non consente un aggiornamento in automatico e comporta anche che ad ogni modifica dell'insieme di regole si metta mano al codice e si provveda a ricompilarlo; però è l'unico modo per poter integrare con una certa flessibilità all'interno di questa struttura le varie regole che possono modellare la realtà esistente.

Appurato come la seconda soluzione, quella 'cablata', sia senza dubbio la più praticabile, rimane da osservare che tale insieme di regole andrà aggiornato ogni qual volta si

proceda all'inserimento di nuove risorse all'interno del VL, e che dunque in tal caso si dovrà procedere alla ricompilazione del sorgente.

In tal caso, infatti, il Lab Master dovrà fare in modo, ogni volta che nel Laboratorio Virtuale venga aggiunta una nuova risorsa che presenti interazioni con qualcuna delle precedenti, di modellare opportunamente tali interazioni all'interno del Resources Interaction Set, accessibile dal Security Manager.

Tale codice, per semplicità, dovrà essere scritto in un comune linguaggio di alto livello (ad esempio Pascal, C, C++ o Java) e dovrà essere ben documentato in modo da favorire l'aggiunta di risorse anche a distanza di tempo e anche da parte di programmatori differenti.

Una breve descrizione dei metodi implementati all'interno del Security Manager per appurare la fidatezza e la pericolosità o meno delle operazioni remote da eseguire sarà presentata dopo il seguente paragrafo che conclude la descrizione formale del Modello di Sicurezza del Laboratorio Virtuale.

12.8 Operazioni nel modello del laboratorio

Come nel modello BLP, anche nel caso del modello del laboratorio bisogna stabilire quali sono le operazioni consentite e quali modifiche comportano sulle strutture dati.

Le operazioni possibili sono le seguenti:

- **Conseguimento dell'accesso**: l'utente chiede di accedere ad una risorsa; viene verificato se la necessaria autorizzazione è presente nella Access Matrix, se il Resource Interaction Set non prevede controindicazioni per l'operazione richiesta e se il Current Blocking Set indica che tale risorsa è bloccata da altre operazioni in corso; nel caso l'operazione sia sicura e la

risorsa sia disponibile si consente l'accesso andandolo a registrare nel Current Access Set, ed eventualmente nel Blocking Set se l'operazione richiede il bloccaggio.

- **Rilascio dell'accesso**: viene rimosso il corrispondente elemento dal Current Access Set e viene registrato il tempo complessivo per cui l'utente ha utilizzato la risorsa all'interno dello User History; se l'operazione bloccava la risorsa bisogna rimuovere il bloccaggio dal Current Blocking Set.
- **Iscrizione di un nuovo utente**: bisogna aggiungere alla Access Matrix una nuova riga corrispondente al nuovo utente, che indichi tutti i diritti che gli devono essere conferiti; viene inoltre creto un nuovo elemento per la User History.
- **Cancellazione di un utente**: deve essere rimossa la riga corrispondente dalla Access Matrix e si deve eliminare il corrispondente elemento della User History; si deve inoltre controllare nel Current Access Set se tale utente ha attualmente accesso a qualche risorsa, ed in tal caso se ne deve forzare il rilascio.
- **Aggiunta di una risorsa al laboratorio**: bisogna aggiungere una nuova colonna alla Access Matrix, valutando in corrispondenza di ogni utente se si vuole fornire o meno l'accesso ad essa; bisogna inoltre inserire nel Resource Interaction Set le nuove regole inerenti la nuova risorsa.
- **Rimozione di una risorsa**: bisogna cancellare la riga corrispondente dalla Access Matrix, e forzarne il rilascio da parte di tutti gli utenti che vi stanno attualmente accedendo sulla base di quanto indicato nel Current Access Set; è utile in questo caso rimuovere dal Resource Interaction Set tutte le regole che riguardavano le variabili della risorsa rimossa.
- **Modifica delle autorizzazioni**: l'amministratore del sistema ha la facoltà, oltre che di iscrivere nuovi utenti, anche di modificare nel tempo le autorizzazioni concesse, modificando i singoli elementi della matrice degli accessi; nel caso in cui voglia negare ad un

utente l’accesso ad una risorsa precedentemente consentita, bisogna forzarne il rilascio controllando gli elementi del Current Access Set.

12.9 Algoritmo del Security Manager

Come detto, il Security Manager del Laboratorio Virtuale ha il compito di verificare, grazie alle varie strutture del Modello di Sicurezza mantenute al suo interno, se la richiesta di operazione remota da parte di un utente su di una risorsa debba essere accettata ed eseguita oppure se essa violi uno dei requisiti di sicurezza, come ad esempio la sicurezza dagli accessi o la sicurezza intrinseca degli impianti.

In particolare, appena ricevuta una richiesta da un utente autenticato, il Security Manager la esegue se essa supera il seguente algoritmo, descritto formalmente per comodità in linguaggio C-like:

```
if (utente autenticato) {     // supponiamo protocollo autenticazione
concluso
      while (true) {          // riceve ciclicamente richieste
            receive (risorsa, comando sulla risorsa);
            if (checkAccess(utente, risorsa) && checkCommand(risorsa,
comando)) {
                  execute(risorsa, comando);
                  send(risultato);
            } else {
                  disconnect(utente);
            }
      }
}
```

In particolare i due metodi di controllo del Security Manager dovranno operare come segue: il primo metodo controlla nella matrice degli accessi e restituisce il corrispondente permesso affermativo o negativo, mentre il successivo è quello che contiene cablato al suo interno il ResourcesInteractionSet:

```
boolean checkAccess(utente, risorsa) {
      return (AccessMatrix(utente, risorsa));   // torna autorizzaz.true o
false
}

boolean checkCommand(risorsa, comando) {
      if (CurrentBlockingSet(risorsa)) return false;  // se la risorsa è
bloccata
      return (ResourcesInteractionSet);   // vede se l'interazione è
permessa
}
```

Le altre strutture dati, invece, ovvero la User History ed il Current Access Set, non servono per verificare la fidatezza delle operazioni in arrivo ma per implementare meccanismi di log e di espulsione dopo un certo tempo mediante un thread separato.

13 - Attacchi alla sicurezza del laboratorio

13.1 Tipologie di attacchi

Come attacchi alla sicurezza del Laboratorio Virtuale si intenderà il tentativo di un intruder di violare uno dei requisiti di sicurezza progettati per il VL.

Le varie forme di attacco possono allora essere catalogate a seconda del requisito soggetto ad attacco:

1. attacchi contro la Outsider Access Security;
2. attacchi contro la Insider Access Security;
3. attacchi contro la Intrinsic Lab Operation Security.

Oltre questi attacchi contro il laboratorio vero e proprio bisognerà prendere in esame anche le seguenti forme di attacco:

1. attacchi contro la riservatezza dell'utente;
2. attacchi contro il sottosistema del Laboratorio Virtuale.

I vari tipi di attacchi saranno discussi nei paragrafi seguenti, in cui si cercherà di valutare se i meccanismi di protezione progettati siano sufficienti a garantire la sicurezza del laboratorio e quali siano i punti deboli del sistema.

13.2 Attacchi contro la Outsider Access Security

Un attacco contro la Outsider Access Security si verifica qualora un utente della rete non iscritto presso il Laboratorio Virtuale tenta di accedere ugualmente.

In generale, qualunque richiesta di operazioni sulle risorse potrà avvenire tramite le primitive della API, ma sarà presa in considerazione dalla rete solo se accompagnata dal riscontro di una autenticazione precedentemente avvenuta.

Allora un attacco di questo tipo può essere perpetrato con due metodologie differenti:

1. l'intruder riesce ad autenticarsi autonomamente fingendosi uno degli utenti iscritti presso il laboratorio; tale caso sarà detto **Cheating Authentication** e sarà analizzato nel prossimo paragrafo;
2. l'intruder si inserisce 'al volo' nella comunicazione di un utente iscritto che si è autenticato, ed invia le proprie richieste di comandi a distanza; tale caso viene detto **Flying Insertion** e sarà analizzato successivamente.

13.3 Cheating Authentication

Per compiere una Cheating Authentication, l'intruder deve conoscere oppure indovinare il nome e la password di uno degli utenti iscritti.

Nel primo caso, è chiaro che se l'intruder **conosce** esattamente tali informazioni per il laboratorio diventa indistinguibile dall'utente autorizzato, e potrà compiere esattamente le stesse operazioni. Tale eventualità è possibile ad esempio nel caso in cui l'intruder sia stato presente durante un precedente accesso di tale utente, e sia riuscito a leggergli la password osservando il movimento delle dita; c'è da osservare come diversi intruder siano specializzati nell'arte di ricavare le password osservando la digitazione.

Uno dei punti cardine per evitare che qualche intruder possa compiere una Cheating Authentication consiste allora nello spronare gli utenti legittimati ad essere molto circospetti durante gli accessi. Per fare ciò bisogna innanzitutto prevedere, come del resto è stato fatto, una forte responsabilizzazione degli utenti riguardo alla possibilità che qualcun altro possa accedere al loro posto; ovvero, bisogna fare in modo che ad essi stessi non convenga una simile eventualità, magari prevedendo delle forme di pagamento proporzionali al tempo di utilizzo del laboratorio.

Con la seconda eventualità, l'intruder riesce ad autenticarsi **indovinando** nome e password di un utente legittimato. Riguardo al nome non è difficile ricavare i criteri con cui il Lab Master attribuisce i nomi di login ad i vari utenti, e possiamo dunque supporre che tutta la segretezza risieda nella password.

Affinché l'intruder riesca dunque ad indovinare la password, deve essergli possibile tentare varie volte utilizzando parole di uso comune; tale tipo di attacco è detto *Attacco con Dizionario*, ed è una tecnica molto potente se fatta attuare in automatico da un apposito programma che tenti ciclicamente di attuare l'autenticazione utilizzando le parole contenute in un apposito file (detto, per l'appunto, *dizionario*).

Per evitare l'attuazione di un attacco con dizionario, possono attuarsi due diverse strategie, una a monte e l'altra a valle. Innanzitutto si potrebbe prevedere un meccanismo di iscrizione degli utenti che li obblighi ad utilizzare come password delle parole sufficientemente lunghe e non di uso comune, costringendoli magari ad inserire nella parola d'ordine utilizzata caratteri minuscoli, maiuscoli e speciali (simboli).

Inoltre la procedura di autenticazione potrebbe essa stessa essere congegnata in modo tale da evitare l'attuazione di un simile attacco: ad esempio potrebbe essere inserito un tempo d'attesa sempre maggiore tra due tentativi falliti di autenticazione, e dopo un certo numero

massimo di tentativi potrebbero essere sospesi del tutto gli accessi e potrebbe essere emesso qualche segnale di avviso per i gestori del laboratorio.

13.4 Flying Insertion

Per compiere una Flying Insertion, l'intruder deve avere la possibilità di intercettare i messaggi in transito sulla rete tra un utente autenticato ed il laboratorio, e deve poterne fabbricare di nuovi con le proprie richieste.

In realtà tale tipo di attacco è molto più difficile da realizzare di quanto potrebbe sembrare, in quanto tutte le informazioni che vengono scambiate tra il laboratorio e l'utente, siano esse basate sul protocollo Request/Reply o su quello Continuous Flow, viaggiano sulla rete in forma cifrata.

In virtù di questo fatto, l'intruder dalle intercettazioni dovrebbe innanzitutto ricavare la chiave di sessione utilizzata, e poi utilizzarla per cifrare le proprie richieste appositamente fabbricate da spedire fraudolentemente al Lab Server.

Grazie però all'opportuno protocollo di autenticazione, per l'intruso è del tutto impossibile ricavare la chiave di sessione se non conosceva precedentemente la password dell'utente; ma se avesse questa informazione, l'intruso potrebbe molto più semplicemente tentare una autenticazione autonoma, ed l'attacco si ridurrebbe al precedente caso di Cheating Authentication.

In definitiva tale attacco è reso praticamente impossibile dall'utilizzo di un protocollo di autenticazione con chiave doppia, che permetterebbe di interpretare correttamente i messaggi intercettati e di fabbricarne opportunamente di nuovi solo conoscendo a priori la password dell'utente.

13.5 Attacchi contro la Insider Access Security

Un attacco contro la Insider Access Security si verifica allorquando un utente iscritto ed autenticato presso il laboratorio invia delle richieste di operazioni per le quali non è autorizzato.

La libertà di inviare le richieste è totale, in quanto il *parameter marshalling* delle primitive del modello Client/Server è noto e documentato, ma come visto tutte le richieste prima di essere messe in atto vengono controllate dai meccanismi di protezione del sistema.

In particolare il Modello di Sicurezza del Laboratorio Virtuale permette di eseguire solo operazioni per le quali è presente un apposito entry all'interno della Access Matrix, e dunque prima di compiere ogni operazione il Security Manager deve accertarsi che l'utente abbia l'autorizzazione per compiere l'operazione richiesta senza che vi sia nessuna possibilità di aggirare questo ostacolo.

Un caso particolare di attacco contro la Insider Access Security potrebbe essere considerato, a seconda della definizione, quello in cui un utente iscritto presso il VL tenti di autenticarsi sotto le sembianze di un altro utente dotato magari di un maggior numero di permessi. In questo caso in realtà è come se tale utente fosse un perfetto intruso per il laboratorio, e vale tutto quanto già detto a proposito degli attacchi contro la Outsider Access Security.

13.6 Attacchi contro la Intrinsic Lab Operation Security

Tali attacchi si verificano quando degli utenti iscritti ed autenticati presso il laboratorio richiedono di effettuare delle operazioni che potrebbero rivelarsi pericolose per gli impianti su cui dovrebbero essere effettuate.

Per garantire che in nessun caso possano generarsi situazioni di pericolo, come già visto, i vari comportamenti degli impianti devono essere modellati all'interno di un modulo software attivo, che abbiamo chiamato Security Manager. Tale modulo deve provvedere a controllare le singole operazioni richieste, andando all'occorrenza anche ad effettuare un rilevamento autonomo dei valori di alcune variabili di stato per verificare se l'interazione tra le varie parti dell'impianto possa portare in stati non sicuri.

Il soddisfacimento della Intrinsic Lab Operation Security, che è sicuramente la principale forma di sicurezza che si richiede per un sistema distribuito come il Laboratorio Virtuale, dipende dunque totalmente dalle modalità con cui è stato programmato il Security Manager. E' dunque necessario che l'implementatore del VL disponga delle completa conoscenza delle leggi fisiche che regolano il funzionamento e le interazioni di tutti gli impianti, e che possa lavorare a stretto contatto con i tecnici specializzati degli impianti stessi.

Come già specificato parlando del Resources Interaction Set, le informazioni riguardanti i limiti di funzionamento degli impianti e le interazioni tra le loro parti vanno opportunamente programmate all'interno del Security Manager.

In particolare il Security Manager dovrà disporre di un opportuno metodo che, ricevendo in ingresso l'operazione richiesta dall'utente, analizzandola ed individuando in essa la risorsa coinvolta ed i parametri di ingresso, sia in grado di stabilire se l'operazione sia fidata o meno, restituendo in tal caso in uscita un valore booleano.

13.7 Attacchi contro la riservatezza dell'utente

Oltre agli attacchi contro il laboratorio vero e proprio, qualche intruso potrebbe cercare di effettuare altri tipi di attacco, ad esempio contro la riservatezza delle informazioni che l'utente scambia sulla rete.

L'utente, come detto nel capitolo 10, per svariati motivi potrebbe desiderare che le informazioni che si trova a scambiare con il Laboratorio Virtuale, ad esempio le richieste di comandi a distanza ed i vari risultati delle elaborazioni, siano mantenute strettamente riservate e non possano in nessun caso essere intercettate ed utilizzate da qualche intruso.

In base al progetto del sistema di sicurezza del Laboratorio Virtuale, la riservatezza di tutte le informazioni in transito sulla rete tra l'utente ed il Lab Server viene assicurata dal protocollo di autenticazione che è stato studiato nel capitolo 11. Come abbiamo più volte visto, tale protocollo si basa sulle più moderne tecniche di cifratura dei dati e garantisce che anche qualora vi fosse stato un intruso in ascolto sin dall'inizio lungo la connessione non potrebbe in nessun caso riuscire a decifrare i dati intercettati.

13.8 Attacchi contro il sottosistema del Laboratorio Virtuale

Anche se il Laboratorio Virtuale di per sé riesce a garantire la sicurezza degli accessi e delle operazioni, non è detto che altrettanto faccia il sottosistema su cui il Lab Server viene ad essere implementato.

Qualunque realizzazione del VL, infatti, dovrebbe consistere di un insieme di programmi e di strutture memorizzati sotto forma di file all'interno di un disco di un computer connesso in rete, che a tal proposito stiamo chiamando *sottosistema*; se tale sottosistema è di per sé non sicuro, ovvero non effettua i necessari controlli su chi tenti di accedere al sistema

dalla rete o anche dalla consolle, è probabile che un intruso riesca ad alterare i file memorizzati sul disco.

La possibilità di modificare i file presenti sul disco consentirebbe all'intruso di distruggere tutta l'implementazione del Laboratorio Virtuale, negando di fatto la prestazione del servizio agli utenti autorizzati (*denial of service*); tale operazione sarebbe estremamente grave però verrebbe immediatamente diagnosticata dal Lab Master che potrebbe ripristinare il sistema con un opportuno back-up.

Tale accesso permetterebbe all'intruso anche di registrarsi autonomamente, cosa che probabilmente verrebbe individuata con notevole ritardo, o addirittura gli permetterebbe di inserire nella procedura di registrazione dei nuovi utenti un *trojan horse* col quale potrebbe venire a conoscenza delle password di tutti i nuovi utenti, cosa che non potrebbe essere diagnosticata in nessun modo.

E' chiaro dunque come in questa sede ci si sia occupato di rendere sicuro il Laboratorio Virtuale così come è stato progettato, ma che altrettanta cura deve essere posta nella scelta di un sottosistema che fornisca adeguate misure di sicurezza per limitare gli accessi al disco e alla memoria della macchina su cui il server del laboratorio debba essere installato.

Ad esempio, sebbene sia possibile installare il Lab Server su di un qualunque PC con Windows 95 (necessario per far girare la Java Virtual Machine su di un PC) sarebbe di gran lunga preferibile utilizzare una molto più sicura workstation Unix (ad esempio una SPARCstation di Sun, per la quale il JDK è stato originariamente sviluppato).

14 - Implementazione del Laboratorio Virtuale

14.1 Schema generale in linguaggio Java

E' ora possibile delineare uno schema elementare di implementazione del Laboratorio Virtuale. Per la realizzazione è opportuno adoperare il **linguaggio Java** che è l'unico che permette di ottenere dal VL tutte le caratteristiche richieste, come ad esempio la *Platform Indipendence* e l'*Upgrading Indipendence* che si erano descritte nel capitolo 1.

In particolare i vantaggi del linguaggio Java risultano notevoli se si tiene presente che la **GUI** potrebbe essere implementata come una *applet*, in modo tale da consentire agli utenti un accesso remoto rapido semplicemente disponendo di un Web browser che sia Java-enabled, come del resto tutti i Web browser di recente rilascio.

Tutte le funzioni offerte dal VL sono realizzate mediante una serie di classi in linguaggio Java; in particolare per l'accesso tramite API tali classi dovranno essere fornite manualmente, ovvero tramite FTP effettuato dall'utente ad ogni richiesta di aggiornamento (per cui delle due proprietà richieste varrà solo la Platform Indipendence, mentre la Upgrading Indipendence viene sacrificata per ottenere una maggiore flessibilità di assemblaggio delle varie risorse).

Della GUI, invece, come detto, sarà il Web browser che si farà carico volta per volta di caricare l'ultima versione aggiornata in tempo reale; tale GUI comprenderà le stesse classi Java della API più altre classi per gestire la applet e l'interfaccia grafica.

14.2 Implicazioni dell'uso di Java sull'architettura

L'utilizzo del linguaggio Java comporta alcune implicazioni sulle scelte di progetto che erano state analizzate a proposito delle varie possibilità di organizzazione dell'architettura del Virtual Lab.

Riguardo alle modalità di comunicazione, sebbene Java disponga delle opportune classi atte ad implementare una **Remote Method Invocation** (*RMI*), che non è altro che una chiamata di procedura remota, si è visto come in realtà non sia possibile prevedere per una RPC le funzionalità atte ad effettuare il monitoraggio remoto di un impianto, e che si renda opportuno un modello Client/Server con gli opportuni protocolli Request/Reply e Continuous Flow.

L'utilizzo di un modello Client/Server progettato verticalmente, a partire dal livello più basso, ovvero i socket, per arrivare a delle chiamate di procedura più semplici che sia possibile, permette anche una notevole flessibilità in quanto tale modello è poco dipendente dagli strumenti specifici messi a disposizione dal linguaggio Java, ed è anche modulare in quanto ogni singola classe può essere facilmente espansa per inglobare delle nuove funzionalità.

Come protocollo da adoperare per la comunicazione tramite socket si è preferito scegliere il TCP, protocollo affidabile con connessione, in modo da rendere la comunicazione tra client e server il più affidabile possibile, consentendo anche la ricezione dei messaggi nel corretto ordine di trasmissione.

Un’altra implicazione dell’utilizzo di Java sull’architettura è che, volendo fornire per tutte le risorse anche delle **GUI** implementate come **applet**, tali interfacce presenteranno tutte le limitazioni delle applet stesse ed in particolare potranno collegarsi tramite socket solamente con l’host da cui sono state scaricate (dunque il Lab Server dovrà girare sullo stesso host del Web server).

Questa limitazione di Java, non aggirabile, è stata introdotta da Sun per preservare al massimo la riservatezza degli utenti che scaricano applet dalla rete col proprio browser; in ogni caso, per il Laboratorio Virtuale, tale limitazione conduce ad alcune scelte obbligate per quanto riguarda l’organizzazione dell’architettura del Sistema Distribuito, ma non limita in nessun altro modo le possibilità di interazione offerte dal laboratorio agli utenti.

Questa limitazione introdotta da Java comporta in definitiva solo che la eventuale replicazione del Lab Server sia impossibile per gli utenti collegati tramite GUI, e che soprattutto il Lab Server debba sempre essere presente a fare da intermediario tra il client e lo slave prescelto per eseguire l’operazione richiesta dall’utente.

Una ultima considerazione va fatta a proposito della realizzazione di accessi selettivi ai **tutorial**, per la qual cosa sarà possibile fare in modo che, dopo che l’utente abbia avuto accesso tramite la GUI visualizzata dal Web browser, il Lab Server costruisca su disco un file HTML di ipertesto contenente i tutorial cui l’utente ha il diritto di accesso, e poi comunichi al Web browser solamente il nome della pagina che gli verrà poi fornita dal Web server. Tale possibilità di interazione tra una applet ed il browser è infatti possibile grazie alla classe `AppletContext` di Java.

14.3 Modalità di connessione con il Lab Server

Per consentire al massimo numero di utenti di accedere contemporaneamente alle possibilità offerte dal Laboratorio Virtuale, il Lab Server è stato organizzato mediante un server concorrente, ovvero viene avviata per ogni client una copia differente del server e tutte le copie possono essere eseguite l'una indipendentemente dall'altra.

Per fare ciò, l'utente si collega inizialmente col ***MultiLabServer***, che è unico per tutti ed il cui ServerSocket è posto in attesa di una connessione TCP su una porta di numero prefissato (nel nostro caso si è scelto ad esempio la porta numero 6666).

Questo MultiLabServer ha come unico compito quello di istanziare per ogni nuova connessione una nuova copia del thread ***SingleLabServer***, assegnarle un'altra porta (stavolta privata) e di comunicarla al client.

In tal modo, mentre l'unico MultiLabServer è in ascolto sempre sulla stessa porta e serve solo per distribuire gli utenti ognuno su una differente porta del server, ognuno di essi avrà a disposizione un proprio SingleLabServer con un numero di porta privato con cui stabilire la connessione ed effettuare in seguito i vari protocolli di Autenticazione e di comunicazione (Request/Reply o Continuous Flow).

14.4 Catalogazione delle classi Java utilizzate

Nel complesso le classi Java implementate possono essere così raggruppate:

1. **classi della API del client**: sono le classi utilizzate dal programmatore per accedere alle risorse del laboratorio direttamente dalle proprie applicazioni;
2. **classi della GUI del client**: sono le classi eventualmente utilizzate dal browser nel caso in cui l'utente voglia accedere al VL tramite la GUI predisposta per la sola configurazione dei parametri;

3. **classi del server**: sono le classi che il Lab Server utilizza per compiere tutte le operazioni che da esso dipendono, come l'iscrizione di nuovi utenti, la autenticazione iniziale, la gestione delle strutture del modello di sicurezza, la gestione delle richieste in arrivo e la spedizione delle risposte ai vari richiedenti;
4. **classi di supporto utilizzate sia dal client che dal server**: ci sono alcune classi di utilità che vengono utilizzate sia dal client che dal server, ad esempio per la cifratura dei dati e per la rappresentazione delle informazioni scambiate lungo la rete.

Le classi appartenenti alle varie categorie saranno particolareggiate nel seguito.

14.5 Classi della API del client

Nel caso in cui l'utente voglia interfacciarsi direttamente con le risorse del laboratorio dovrà utilizzare opportunamente i metodi di due opportune classi:

1. **`LabClient`**: è la classe che incorpora i metodi per inviare al server i dati utili alla autenticazione dell'utente, per spedire ad esso le proprie richieste e per ricevere la successiva risposta.
2. **`LabResources`**: è la classe che fornisce al client le informazioni riguardo alle risorse attualmente disponibili ed il formato in cui devono essere inviate le richieste di comandi sulle varie risorse.

Lo sviluppatore oltre queste due classi dovrà tener presente che esistono altre classi da utilizzare tra le classi di supporto; per lui è necessario solo avere conoscenza delle modalità

con cui utilizzare i metodi di tali classi (queste due più quelle di supporto) per potersi interfacciare con le risorse del laboratorio.

C'è da notare che la prima cosa che deve fare l'utente per poter successivamente interagire con il laboratorio è invocare nella maniera opportuna il metodo di autenticazione del Lab Client; in caso contrario il Lab Server scarterà a priori tutti i messaggi ricevuti da un utente non autenticato.

L'invocazione del metodo di autenticazione da parte dell'applicazione sviluppata da un utente può avveniere con diverse modalità, che il Lab Master non ha nessuna possibilità di impedire in quanto l'accesso al laboratorio è trasparente rispetto al corretto utilizzo della API:

1. lo sviluppatore può decidere di incorporare nella propria applicazione la stessa procedura seguita all'atto dell'esecuzione della GUI da parte del browser: ovvero, può fare in modo che il proprio programma ad ogni esecuzione richieda che l'utilizzatore digiti le proprie informazioni personali da utilizzare per l'autenticazione;
2. lo sviluppatore può fornire il proprio programma di una opportuna installazione, in base alla quale ogni utilizzatore la prima volta che esegue il programma sulla propria macchina setta il proprio nome e la propria password in base ai quali dovrà essere di volta in volta compiuta l'autenticazione; tali informazioni verranno così ad essere cifrate e memorizzate su disco, e non verranno quindi più richieste in occasione delle successive esecuzioni; questa soluzione è parzialmente pericolosa in quanto chiunque si trovi ad avere accesso al computer su cui il programma è stato installato potrà accedere liberamente alle risorse messe a disposizione di chi ha provveduto ad installare il programma;
3. lo sviluppatore prevede di utilizzare il propgramma sviluppato solo per fini personali ed allora cabla i propri dati personali (cifrati) all'interno del proprio programma; tale approccio presenta lo stesso svantaggio della soluzione precedente.

Probabilmente la scelta migliore è la prima, ovvero l'utente può effettuare di volta in volta il login ad ogni connessione, il che può spingere correttamente tutti gli utenti a modificare le proprie password con una certa frequenza.

L'unico modo per cercare di limitare gli accessi impropri al VL consiste allora nel cercare di responsabilizzare gli utenti registrati, ad esempio realizzando effettivamente dei meccanismi di controllo dell'effettivo utilizzo delle risorse del laboratorio da parte di ognuno di essi; in tal modo ogni singolo sviluppatore sarebbe spronato o a mantenere il proprio programma il più sicuro possibile oppure a vigilare con maggiore attenzione su chi possa accedere al proprio computer.

14.6 Classi della GUI del client

Qualunque utente può inizialmente accedere col proprio Web browser alla pagina di presentazione del laboratorio, che si trova nel file `VirtualLab.html`. Da questa pagina viene eseguita una applet che richiede all'utente di digitare le proprie informazioni personali (ovvero lo User Name e la Password fornitegli all'atto della registrazione) e stabilisce una connessione con il Lab Server.

Se l'utente viene autenticato correttamente e dispone delle autorizzazioni necessarie, l'applet fornisce una GUI tramite cui poter interagire con le sole interfacce relative alle risorse cui ha diritto ad accedere. In particolare la GUI presenta una bottoniera che presenta tutti i nomi delle risorse accessibili; cliccando su uno dei bottoni viene presentata all'utente l'interfaccia di ingresso della risorsa selezionata; al ritorno dalla esecuzione del comando sul server, il risultato viene presentato all'utente nell'interfaccia grafica di uscita.

Per realizzare una GUI sono necessarie le stesse classi richieste per la API, più altre classi per gestire l'esecuzione della applet e delle interfacce grafiche:

1. **`VirtualLab`**: è l'applet collegata alla pagina HTML da cui viene avviata l'esecuzione; è quella che coordina l'esecuzione di tutte la altre parti, provvedendo ad esempio ad eseguire la richiesta dei dati da digitare per l'autenticazione, ad avviare il protocollo ed a visualizzare sullo schermo le varie finestre.
2. **`AnimationThread`**: è il thread che gestisce una piccola animazione nello spazio dell'applet mentre l'utente attende che venga eseguito il login; è implementato come thread per permettere la sua esecuzione in parallelo alle altre attività del laboratorio.
3. **`LoginDialog`**: è la finestra di dialogo in cui viene chiesto all'utente di digitare il proprio User Name e la propria Password.
4. **`LabResourceFrame`**: è la finestra in cui viene presentata all'utente la bottoniera con i nomi di tutte le risorse accessibili.
5. **`InputFrame`**: è la finestra in cui viene rappresentata l'interfaccia di ingresso della risorsa prescelta.
6. **`OutputFrame`**: è la finestra in cui vengono rappresentati in forma grafica i risultati dell'esecuzione remota.

14.7 Classi del server

Le classi utilizzate dal processo server, a parte le classi di supporto, sono le seguenti:

1. **`LabServer`**: è l'applicazione stand-alone che permette di avviare il server iterativo che deve essere sempre attivo in background sull'host su cui gira il Web server.

2. **`MultiLabServer`**: è il thread che implementa il server iterativo che si pone in ascolto, mediante la classe `ServerSocket` di Java, su di una porta di numero prefissato (nel nostro caso 6666), ed appena riceve una richiesta di connessione TCP seleziona un'altra porta privata (una qualunque di quelle libere) il cui numero viene comunicato al Lab Client e ad una nuova istanza della classe seguente. In ogni momento sul server è attiva una e solo una istanza di questa classe.
3. **`SingleLabServer`**: è il thread che si pone in ascolto di richieste sulla porta privata fornitagli dal thread precedente, e si occupa di gestire tutte le operazioni intraprese dal Lab Server, dall'autenticazione al conferimento delle autorizzazioni, dalla gestione delle informazioni memorizzate su file alla consultazione del security manager, fino allo smistamento dei comandi tra i vari slave e la trasmissione delle risposte, sia per il protocollo Request/Reply che per quello Continuous Flow. In ogni momento sul server è attiva una istanza di questa classe per ognuno degli utenti connessi.
4. **`LabSecurityManager`**: è la classe che si occupa di gestire tutto il Modello di Sicurezza del Virtual Lab, memorizzando ed aggiornando su file le strutture dati non volatili, come la Access Matrix e la User History, tenendo traccia in memoria delle strutture che vanno reinizializzate ad ogni riaccensione del sistema, come il Current Access Set ed il Current Blocking Set, e mantenendo cablate al suo interno in un apposito metodo il Resources Interaction Set.
5. **`FileManager`**: è la classe tramite la quale la classe precedente gestisce l'accesso ai file su disco in cui il Lab Server memorizza le strutture dati non volatili del Modello di Sicurezza, ovvero la Access Matrix e la User History; prevede diversi metodi per facilitare la lettura e la scrittura di interi record di dati sotto forma di stringhe di caratteri.

6. **`ExecutionThread`**: è il thread che permette di eseguire un qualunque programma, come ad esempio un simulatore, scritto in un qualunque linguaggio, e di agganciarsi all'output da esso prodotto.

Oltre queste classi per gestire il server vero e proprio, il Lab Master ha a disposizione una serie di comandi di gestione implementati nelle seguenti classi:

1. **`addUser`**: viene utilizzato per iscrivere nuovi utenti al VL, digitando il nome, la password ed i diritti conferiti sulle risorse esistenti; memorizza tali informazioni nel file `data/access` che funge da Access Matrix.
2. **`listUsers`**: elenca tutti gli utenti e le loro autorizzazioni leggendoli dalla Access Matrix.
3. **`listResources`**: elenca tutte le risorse attualmente disponibili.

14.8 Classi di supporto

Sono infine necessarie ulteriori classi che devono essere utilizzate sia dai lati server che client del Laboratorio Virtuale per la cifratura e la gestione del formato dei dati:

1. **`CryptographyLibrary`**: tale classe contiene l'interfaccia per i metodi delle classi DES e RSA per effettuare la cifratura dei dati mediante gli appositi algoritmi; in tal modo è possibile dichiarare una sola classe che presenta solo i semplici metodi pubblici dei due algoritmi. Entrambi gli algoritmi vengono utilizzati tanto dal server quanto dal client per scambiarsi in maniera cifrata le informazioni inerenti i dati per l'autenticazione, le richieste di comandi a distanza ed i risultati ottenuti.

2. **DES**: contiene l'algoritmo di cifratura DES per blocchi di 64 bit, nonché anche i metodi per cifrare blocchi più lunghi mediante i Modi di Operazione ECB (Electronic Codebook) e CBC (Cipher Block Chaining).
3. **RSA**: contiene i metodi per applicare l'algoritmo di cifratura RSA, che richiede anche il test di primalità di Rabin-Miller dei numeri primi generati e l'Algoritmo di Euclide Esteso per ricavare i massimi comuni divisori e gli inversi moltiplicativi.
4. **`DataManager`**: questa classe si è resa necessaria per manipolare correttamente i dati che vengono scambiati tramite socket tra il client ed il server; in particolare, un metodo presente provvede ad impacchettare degli array di stringhe sotto forma di una unica sequenza di byte, ed un altro metodo compie il procedimento inverso. Poiché l'unica forma di dati che può essere trasmessa all'interno di un socket sono delle semplici sequenze di byte, questa classe consente di creare un formalismo tramite cui da un messaggio ricevuto possono essere estratte opportunamente delle stringhe nello stesso formato in cui erano state trasmesse, in modo da esemplificare nella comunicazione la suddivisione tra il comando da eseguire, i parametri da passare ed altre informazioni di controllo.

Il problema del formato dei messaggi che vengono scambiati tra il `LabClient` ed il `SingleLabServer` potrebbe sembrare banale, ma in realtà provvedere una volta per tutte a stabilire un formalismo per la formattazione dei messaggi che vengono scambiati comporta in seguito un notevole risparmio di tempo, poiché chi riceve un messaggio per interpretarlo correttamente non dovrà più scandire uno alla volta i vari byte ma si troverà un comodo array di stringhe già pronto per essere testato.

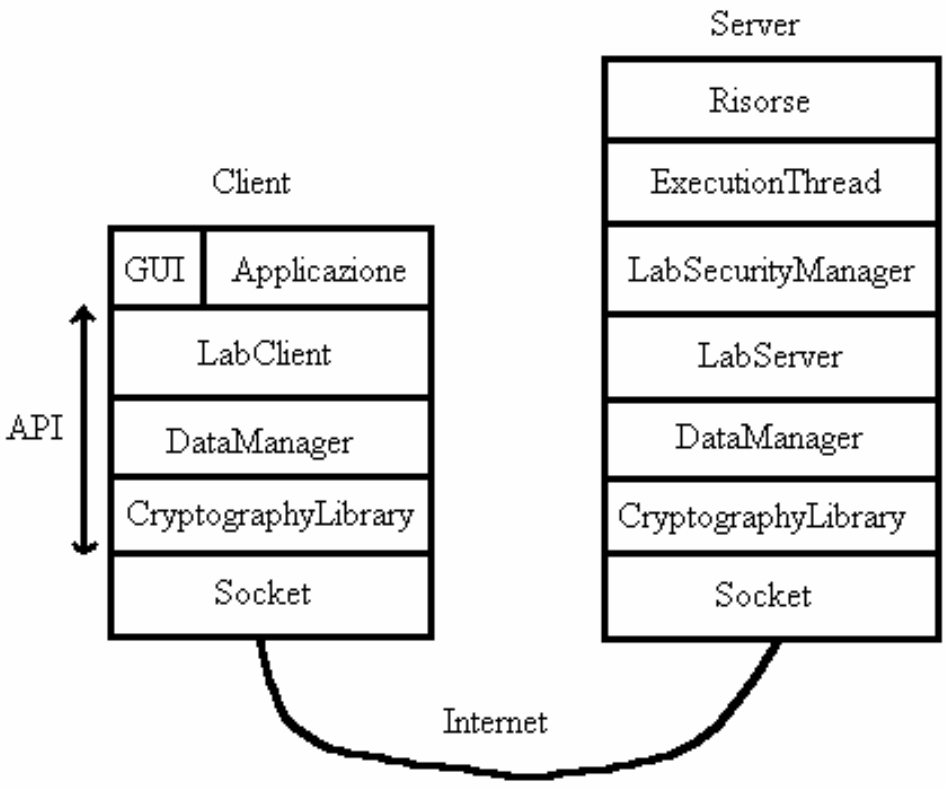
I metodi che servono a impacchettare un array di stringhe in un array di byte, e viceversa, sono `byte[] DataManager.packStringsIntoBytes(String in[])` e `String[] DataManager.unpackStringsFromBytes(byte in[])`.

L’algoritmo sviluppato per impacchettare una serie di stringhe all’interno di una sequenza di byte incorporando in essa pure informazioni inerenti il numero e la lunghezza dell’array è del tutto originale ed è descritto dettagliatamente nel commento presente nel codice sorgente, che viene di seguito riportato:

```
/* Quando voglio spedire delle stringhe nel socket,
esse non vengono trasformate direttamente in bytes
ma viene creata una struttura del seguente tipo:

      bytes[0] = n = numero di stringhe;
      bytes[1]...bytes[n] = lunghezza di ognuna delle n stringhe;
      bytes[n+1]...bytes[n+bytes[1]] = stringa 0;
      bytes[n+bytes[1]+1]...bytes[n+bytes[1]+bytes[2]] = stringa 1;
      ...
      bytes[n+bytes[1]+...+bytes[n-1]+1]...bytes[n+bytes[1]+..+bytes[n]]
                  = stringa n-1

In tal modo la destinazione può facilmente compiere
il procedimento inverso per ricavare le stringhe.

Nel caso in cui i dati debbano essere cifrati, essi alla sorgente
vanno prima impacchettati in array di byte, poi cifrati e spediti,
e alla destinazione vanno decifrati e ricomposti come stringhe.

N.B. Si vede che deve valere la proprietà:
      bytes.length == bytes[0]+bytes[1]+...+bytes[n]+1
In pratica però mi accontento che sia >= , perché la cifratura
potrebbe richiedere l'aggiunta di caratteri di riempimento. */
```

Da notare che tutti i messaggi che a regime (ovvero dopo lo scambio della chiave di sessione) vengono scambiati tra client e server, sotto forma di array di stringhe, devono essere prima convertiti in sequenze di byte tramite il `Data Manager`, poi cifrati con i metodi DES della `CryptographyLibrary`, successivamente spediti tramite socket, decifrati con lo stesso algoritmo DES, e ricomposti sotto forma di array di stringhe grazie nuovamente al Data Manager.

Questo procedimento consente di compiere una prima analisi sulla fidatezza del messaggio ricevuto, in quanto un messaggio decifrato con la chiave di sessione prevista per quell'utente che non rispetti la formattazione prevista dall'algoritmo verrà immediatamente scartato.



15 - La API ed il formato delle comunicazioni

15.1 Metodi della API

In definitiva la API disponibile per il programmatore si compone dei metodi delle seguenti classi:

1. **`LabClient`**: comprende i metodi per l'autenticazione, per inviare i messaggi al server e per ricevere le risposte.
2. **`LabResources`**: i metodi presenti consentono di conoscere il numero ed il nome delle varie risorse rese disponibili dal VL, nonché per ottenere le informazioni riguardo alla modalità con la quale debba essere configurato il messaggio da inviare al server (nome del comando da eseguire, numero ed ordine dei parametri, valori possibili, ecc.).
3. **`DataManager`**: come detto permette di trasformare un array di stringhe (contenenti di volta in volta i messaggi per le autenticazioni, i comandi da inviare al server e le relative risposte) in un array di byte opportunamente formattato; sono quindi presenti metodi per compiere la trasformazione in un senso e nell'altro.
4. **`CryptographyLibrary`**: i suoi due metodi permettono di cifrare delle sequenze di byte utilizzando di volta in volta gli algoritmi DES o RSA.

In realtà si è cercato di rendere quanto più semplice possibile l'implementazione di nuove applicazioni da parte dello sviluppatore utilizzando la API, per cui tutta la complessità è stata nascosta allo sviluppatore che necessita solo di conoscere tre metodi della classe LabClient, sufficienti a supportare il protocollo di autenticazione ed i due protocolli di comunicazione.

In definitiva, dopo avere appreso tramite la classe `LabResources` le modalità di *parameter marshalling* per effettuare richieste alle varie risorse, tutto ciò di cui si necessita è invocare opportunamente i seguenti metodi della classe LabClient:

1. **`String[] LabClient.Authentication(String nameAndCryptedPassword[])`** : permette all'utente di autenticarsi inviando il proprio user name e la propria password (già cifrata) e di ricevere in risposta la conferma delle autorizzazioni per l'accesso alle varie risorse.
2. **`String[] LabClient.RequestReply(String command[])`** : permette all'utente già autenticato di inviare una richiesta di comando a distanza e di ricevere in risposta il risultato della elaborazione.
3. **`String[] LabClient.ContinuousFlow(String command[])`** : permette all'utente già autenticato di effettuare una richiesta e di iniziare a ricevere periodicamente i dati forniti dal server. Per mantenere la coerenza semantica delle chiamate di procedura, il programmatore dovrà invocare tale metodo ciclicamente, anche se nella pratica solo alla prima invocazione verrà inviata la corrispondente richiesta al Lab Server. In tal modo l'utente, pur senza appesantire la rete ed il server con una serie di richieste identiche, potrà ciclicamente leggere il valore delle variabili che ha richiesto di monitorare. Così tutta la complessità viene nascosta anche allo sviluppatore che utilizza la API.

Allora, per ottenere l'accesso richiesto, ogni utente deve innanzitutto invocare il primo metodo per autenticarsi, e successivamente può richiedere tutte le operazioni che desidera svolgere sulle varie risorse.

C'è da notare che tutti e tre i metodi della API accettano come parametri un array di stringhe e restituiscono come risultato un altro array di stringhe, in modo tale da rendere per l'utente il più semplice possibile la gestione di alto livello delle informazioni scambiate con il laboratorio.

Una ultima considerazione riguarda il protocollo Continuous Flow, per sfruttare il quale l'utente deve invocare ciclicamente l'opportuno metodo per ottenere in tempo reale i dati richiesti; in realtà tale metodo serve a mascherare che al livello sottostante la richiesta viene inviata solo la prima volta che il metodo è invocato, e la successiva invocazione ciclica (mediante passaggio dei medesimi parametri) serve solamente a garantire che l'utente possa acquisire in tempo reale i dati richiesti mediante i parametri di ritorno mantenendo la struttura logica di una semplice chiamata di procedura.

15.2 Formato delle comunicazioni Client/Server a livello utente

Come detto, tutta la complessità della comunicazione è nascosta all'utente ed è inglobata all'interno delle varie classi della API. Lo sviluppatore deve dunque limitarsi ad utilizzare i tre metodi sopra esposti settando opportunamente il valore degli array di stringhe.

Nel caso dell'autenticazione, l'array da inviare conterrà solamente due elementi, ovvero il nome dell'utente e la password già cifrata; in risposta l'utente otterrà una sola stringa, composta solo di caratteri '`0`' e '`1`', che rappresenteranno le autorizzazioni concesse o meno dal Lab Server all'utente sulle varie risorse; tale stringa potrà essere facilmente convertita

dall'utente in un array di boolean mediante il metodo `boolean[] DataManager.stringToBooleans(String in)`. L'ordine di interpretazione di tali simboli sulle varie risorse è quello che si desume dalla classe `LabResources`, in cui in un array sono contenuti i nomi delle risorse attualmente disponibili.

Per quanto riguarda invece i protocolli Request/Reply e Continuous Flow, l'array di stringhe in ingresso dovrà contenere in prima posizione il nome della risorsa su cui si chiede di interagire, e nelle stringhe seguenti dovranno essere passati i parametri relativi all'interazione richiesta (specificati all'interno della classe `LabResources`). L'array di stringhe ottenuto in risposta dipenderà sempre dalla singola risorsa (le modalità sono descritte sempre in `LabResources`), ma in generale per una interazione con un tool di simulazione saranno le singole righe che l'esecuzione in locale della stessa simulazione produce normalmente sullo standard output.

15.3 Formato delle comunicazioni Client/Server sulla rete

Come detto, uno degli scopi principali della API è quello di nascondere all'utente la complessità degli strati sottostanti, in modo da rendere il più semplice possibile l'effettuazione delle richieste di esecuzione di comandi a distanza; per fare ciò si è scelto di adoperare per l'utente, come formato dei dati in ingresso ed in uscita dai tre metodi di interfaccia (autenticazione, protocolli Request/Reply e Continuous Flow) semplicemente degli array di stringhe già formattati.

In realtà tutte le interazioni tra l'utente ed il laboratorio viaggiano sulla rete sotto forma di flussi di byte senza alcuna formattazione e per di più cifrati. Allora le due classi che stanno ai due lati della connessione sulla rete (tramite socket), cioé il SingleLabServer ed il LabClient,

utilizzano al loro interno vari metodi delle classi `DataManager` e `CryptographyLibrary` per operare le trasformazioni opportune.

Tutti i messaggi vanno dapprima formattati mediante l'algorimo di marshalling presente all'interno della classe `DataManager`, di cui si è già discusso in questo capitolo, con i due metodi `packStringsIntoBytes()` e `unpackStringsFromBytes()`. Successivamente i byte possono essere cifrati con i metodi opportuni (che di norma operano sempre su byte).

Nei casi in cui la cifratura richieda il preventivo riempimento (*padding*) con byte aggiuntivi per raggiungere delle misure prefissate (ad esempio l'arrotondamento per eccessi al blocco di 8 byte superiore per il DES), tali byte possono essere aggiunti casualmente, in quanto dall'altra parte della connessione, dopo la decifrazione, il metodo `unpackStringsFromBytes()` provvederà a scartare gli ultimi byte in eccesso.

C'è da notare che il marshalling della classe `DataManager` di array di stringhe in array di byte si rende necessario in quanto i possibili messaggi che possono essere scambiati sulla rete possono avere dimensioni e formati tra loro diversissimi; però introducono una piccola ridondanza che in alcune situazioni si vorrebbe evitare.

A tal proposito, il metodo `LabClient.Authentication()` provvede, ad un certo punto del protocollo di autenticazione, ad aggirare tale formattazione e a spedire direttamente l'array di byte interessato, di lunghezza nota. Questo capita ad esempio per la trasmissione delle chiavi pubbliche, private e di sessione, nonché per la spedizione del ticket identificativo inviato ad ogni utente. In tal modo si velocizza il procedimento di trasmissione/ricezione (già piuttosto lento per via dell'utilizzo dell'RSA nella sola fase di connessione) e si elimina ogni ridondanza da fasi critiche del collegamento.

16 - Sicurezza e prestazioni con Java

16.1 Aspetti della sicurezza

Come già visto in generale a proposito di un generico Laboriatorio Virtuale, i requisiti di sicurezza per un sistema distribuito così organizzato riguardano la sicurezza dagli accessi esterni ed interni nonché la sicurezza intrinseca degli impianti. In più bisogna tutelare la riservatezza degli utenti, che presumibilmente pagano per avere accesso al laboratorio e non vogliono che i risultati delle operazioni effettuate sul Virtual Lab possano essere intercettate da altri utenti.

Come detto, tutti questi requisiti possono essere ottenuti mediante un protocollo di autenticazione, la cifratura dei dati e l'applicazione del Modello di Sicurezza.

16.2 Protocollo di Autenticazione in Java

Il protocollo di autenticazione implementato in Java è lo stesso che era stato delineato in generale nel capitolo 11. In particolare bisogna confermare l'utilizzo di RSA e DES come algoritmi di cifratura a chiave pubblica (per lo scambio della chiave) e simmetrico (per il prosieguo della comuniucazione).

La gestione del protocollo di autenticazione nelle due parti interessate alla comunicazione avviene all'interno dei metodi `LabClient.Authentication()` e `SingleLabServer.run()`.

Per particolareggiare l'implementazione del protocollo, si osserverà solamente che la chiave di sessione generata dal client dovrà essere un numero intero a 64 bit, ovvero un `long` casuale di Java generato mediante il metodo `Math.random()`, e che per assegnare un identificativo (*ticket*) il server lo sceglierà ugualmente casuale con lo stesso metodo.

C'è da osservare come tale metodo `random()` inizializzi in realtà il proprio algoritmo di generazione di numeri casuali sulla base dell'orario attuale misurato in millisecondi, e che dunque non è possibile in alcun modo prevedere quale sarà il prossimo numero generato.

Per autenticare l'utente, il Lab Server si appoggia al proprio `LabSecurityManager`, che a sua volta accede al file delle password memorizzato su disco tramite la classe `FileManager`. In tal modo viene verificata la corrispondenza tra il nome e la password digitati dall'utente e quelli memorizzati sul file (la password è sempre presente solo in forma cifrata per ridurre al minimo il rischio di intercettazioni).

Il `LabSecurityManager` restituisce dunque solo per l'utente autenticato una stringa di '0' e '1' rappresentanti i permessi di cui l'utente dispone sulle varie risorse; tali permessi servono solo all'utente per conoscenza, in quanto le sue successive richieste verranno di volta in volta controllate singolarmente.

Il protocollo di autenticazione serve in definitiva per proteggere il laboratorio da eventuali attacchi alla External Access Security, in quanto un utente non registrato presso il Virtual Lab non potrà essere in alcun modo autenticato e dunque non potrà in nessun modo inviare richieste al Lab Server.

16.3 Cifratura dei dati

Gli algoritmi di cifratura adoperati per le comunicazioni col Laboratorio Virtuale sono il DES e RSA, che oltre a garantire una elevata affidabilità sono anche gli standard rispettivamente per gli algoritmi simmetrici e quelli a chiave pubblica.

Entrambi presentano le loro difficoltà di implementazione: il DES per l'onere di dover copiare manualmente lunghissimi array di numeri e per la necessità di accompagnare l'algoritmo vero e proprio con altri per implementare i vari Modi di Operazione ECB e CBC per dati più lunghi di 64 bit, l'RSA per la necessità di supportare l'algoritmo di cifratura e decifratura (molto semplice) con vari altri algoritmi di utilità, quali il test di primalità di Rabin-Miller e l'Algoritmo di Euclide Esteso.

Mentre però cifrare dei dati con il DES è una operazione velocissima, altrettanto non si può dire per l'RSA, non solo per quanto riguarda la cifratura e la decifrazione ma anche per la iniziale generazione della chiave pubblica `(n,e)` e di quella privata `(n,d)`.

Fortunatamente l'RSA è richiesto solo all'interno del protocollo di autenticazione, dunque ogni utente si troverà ad attendere qualche secondo in più del dovuto solo al momento del login, quando invece per tutto il resto della comunicazione verrà adoperato il velocissimo algoritmo DES che non introduce ritardi sensibili nella elaborazione.

C'è da notare come la scelta della dimensione del numero '`n`' all'interno dell'algoritmo RSA sia l'elemento che vada scelto come compromesso tra la sicurezza e la velocità dell'algoritmo, nel senso che tanto più è grande e tanto più l'algoritmo sarà sicuro e lento. Nel caso dell'implementazione che è stata realizzata, si è scelto di fare in modo da mantenere tale numero compreso tra 2^{16} e 2^{24}, in modo tale da poter sempre cifrare blocchi di ingresso della dimensione (limitata dal valore inferiore) di 3 byte in blocchi cifrati in uscita della dimensione (limitata dal valore superiore) di 3 byte. Con tali valori, sufficientemente elevati per la sicurezza dell'algoritmo, il protocollo di autenticazione richiede per l'utente una attesa di 1-2 minuti.

Invece a regime tutte le informazioni che vengono scambiate tramite i protocolli Request/Reply e Continuous Flow vengono cifrate con l'algoritmo DES usato con il Modo di Operazione CBC, che è molto più sicuro del modo ECB e molto meno complesso dei modi OFB e CFB.

In definitiva la cifratura dei dati, al di fuori del protocollo di autenticazione, serve soprattutto a garantire la riservatezza dell'utente ma anche a rendere più forte la sicurezza dagli accessi, in quanto ogni richiesta di esecuzione remota dovrà essere cifrata con la chiave opportuna.

16.4 Il Modello di Sicurezza

Dopo avere garantito la External Access Security (tramite il protocollo di autenticazione) e la riservatezza dell'utente (tramite la cifratura dei dati) restano da garantire la Internal Access Security e la Intrinsic Lab Operation Security: di questo si occupa il Modello di Sicurezza, che provvede anche ad effettuare il log delle attività svolte dagli utenti sulle varie risorse.

Il Modello di Sicurezza è implementato all'interno della classe LabSecurityManager. In particolare il metodo `boolean` **`checkAccess(String userName, String resourceName)`** indica se l'utente ha il permesso di accedere alla risorsa indicata (in realtà la classe utilizza `FileManager` per accedere alla **Access Matrix** che è memorizzata su disco all'interno del file delle password). Questo metodo è invocato dal `SingleLabServer` per ogni richiesta, e garantisce quindi il rispetto della Internal Access Security in quanto ogni singola richiesta di ogni utente autenticato deve superare questo controllo prima di essere eseguita.

Invece l'altro metodo boolean **`checkCommand(String commands[])`** serve a verificare che il singolo comando possa essere eseguito senza problemi: in particolare per i simulatori tale metodo si limiterà a verificare che i valori dei parametri passati siano in linea con i limiti statici imposti per i vari tool, mentre per gli impianti reali tale metodo potrà verificare dinamicamente se il valore dei parametri possa causare danni alla Intrinsic Lab Operation Security provvedendo ad effettuare dei rilevamenti autonomi sugli impianti interessati. In tal metodo potranno dunque essere cablate col dettaglio voluto anche tutte le interazioni tra i vari impianti (interazioni che nel Modello di Sicurezza generico andavano sotto il nome di **Resources Interaction Set**).

Entrambi questi metodi vengono invocati dal `SingleLabServer` prima di passare il comando richiesto al modulo che si incarica di effettuare l'esecuzione vera e propria, ovvero l'`ExecutionThread`. In tal modo si garantiscono sia la Internal Access Security quanto la Intrinsic Lab Operation Security.

Oltre alla Access Matrix, che come detto è contenuta all'interno del file delle password, ed al Resources Interaction Set, che è implementato all'interno del metodo `checkCommand`, le altre strutture dati richieste sono la **User History**, che dovrà essere scritta su file per registrare tutte le attività svolte da ogni utente (attività di *log*), nonché il **Current Access Set** ed il **Current Blocking Set**.

Queste ultime due strutture non hanno bisogno di essere memorizzate su file in quanto vengono continuamente modificate e devono essere azzerate ad ogni riavvio del sistema, per cui possono essere facilmente mantenute in memoria in apposite variabili della classe LabSecurityManager.

16.5 Problematiche di implementazione della sicurezza

Come detto, i principali problemi nell'implementazione del protocollo di autenticazione sono sorti per via dell'algoritmo di cifratura RSA, che per funzionare richiede diversi strumenti matematici che, seppur reperibili su alcuni testi di Teoria dei Numeri, sono pur sempre di esecuzione molto lenta, e rendono talvolta estenuante l'attesa in fase di autenticazione.

Di questi algoritmi viene presentata l'implementazione così come sarà utilizzata all'interno della classe RSA del Laboratorio Virtuale.

Il primo algoritmo richiesto è l'algoritmo di Euclide, necessario per trovare il Massimo Comune Divisore (*gcd* sui testi) tra due numeri interi:

```
// Algoritmo di Euclide per trovare il Massimo Comune Divisore
// (gcd = greatest common divisor) di due numeri interi
public long Euclid(long d, long f) {
	long X = f;
	long Y = d;
	long R;

	while (true) {
		if (Y==0) return X;
		R = X % Y;
		X = Y;
		Y = R;
	}
}
```

In realtà per RSA non è sufficiente ricavare il solo MCD, ma serve anche l'inverso moltiplicativo (esiste quando MCD=1) ed in tal caso bisogna applicare l'apposito algoritmo esteso di Euclide, qui riportato:

```
// Algoritmo Esteso di Euclide per trovare il MCD (in long[0])
// e se possibile (MCD=1) anche l'inverso moltiplicativo (long[1])
// ( se l'inverso non esiste torna long[1]=0 )
public long[] ExtendedEuclid(long d, long f) {
	long out[] = { 0, 0 };
	long X1,X2,X3,Y1,Y2,Y3,T1,T2,T3,Q;

	X1=1; X2=0; X3=f; Y1=0; Y2=1; Y3=d;

	while (true) {
		if (Y3==0) {
			out[0] = X3;		// MCD
			return out;		// torna { MCD, 0 }
		}
		if (Y3==1) {
			out[0] = Y3;		// MCD
			if (Y2>0 && (((d*Y2)%f)==1))
				out[1] = Y2;	// inverso (inv d mod f)
			else
				out[1] = 0;	// per la Legge di Murphy
			return out;		// torna { MCD=1, inverso }
		}
		Q = X3 / Y3;		// prendo il quoziente
		T1=X1-Q*Y1; T2=X2-Q*Y2; T3=X3-Q*Y3;
		X1=Y1; X2=Y2; X3=Y3;
		Y1=T1; Y2=T2; Y3=T3;
	}
}
```

Un altro strumento purtroppo necessario è il test di primalità di Rabin Miller, che indica quando un numero intero ha una elevata probabilità di essere un numero primo; questo test si rende necessario in quanto per l'RSA si h la necessità di generare casulamente dei numeri di dimensioni molto grandi e verificare se siano primi o meno.

Da notare che il test è probabilistico, per cui più volte lo si esegue e più probabile sarà che, se il numero supera tutti i test, il numero testato sia primo; ovviamente appena un numero non supera un test si potrà direttamente scartarlo per generarne un altro in quanto sarà probabilmente non primo.

Solitamente il test viene ripetuto 5 volte [SCHN]:

```
	// Il test di primalita' di Rabin Miller dice se e' probabile che
	// l'argomento sia un numero primo (esegue il test singolo 5 volte)
	public boolean RabinMiller(long p) {
		return RabinMiller(p, 5);
	}

	// Il test di primalita' di Rabin Miller dice se e' probabile che
	// l'argomento sia un numero primo (esegue test singolo 'times'
volte)
	public boolean RabinMiller(long p, int times) {
		long test[] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
		long a,b,m;

		// Elimino le estrazioni errate
		if (p<=1) return false;

		// Se e' divisibile per uno dei numeri di test non e' primo
		for (int x=0; x<test.length; x++) {
			if ((p%test[x])==0) return false;
```

```
        }

        // 'b' e' l'exp della max pot di 2 che divide p-1
        long temp = p-1;
        b = 0;
        while(true) {
              if (!myDM.dispari(temp)) {
                    b++;
                    temp = temp/2;
              } else {
                    break;
              }
        }

        // 'm' e' tale che p=1+pow(2,b)*m
        m = (long)((p-1)/(long)Math.pow(2,b));

        // E' probabilmente primo se supera il test 'times' volte
        for (int i=0; i<times; i++) {

              // Genera un numero casuale a<p
              do {
                    a = (long)(Math.random()*(double)p);
              } while(a>=p || a==0);

              if(!singleRabinMiller(p,a,b,m)) return false;
        }
        return true;
  }

  // Procedura usata per un singolo test di primalita'
  // (la procedura pubblica chiama questa 5 o n volte)
```

```
	private boolean singleRabinMiller(long p, long a, long b, long m) {
		long j = 0;
		long z = (((long)Math.pow(a,m)) % p);

		if (z==1 || z==p-1) return true;    // questo test e' superato

		while (true) {
			if (j>0 && z==1) return false;      // di sicuro non e'
primo
			j++;
			if (j<b && z!=p-1) {
				z = ((z*z) % p);
			} else {
				if (z==p-1) return true;      // questo test e'
superato
				if (j==b && z!=p-1) return false;
				break;
			}
		}
		return false;
	}
```

16.6 Prestazioni del Virtual Lab implementato

Lo scopo fondamentale del Virtual Lab è quello di offrire agli utenti un accesso tramite Internet a risorse remote, siano esse risorse software, come simulatori e tool di valutazione automatica, oppure reali impianti industriali. Dunque non è possibile compiere un paragone tra come sarebbe l'accesso alle risorse tramite il Virtual Lab rispetto a come sarebbe senza, in quanto si è venuta a creare una possibilità di accesso prima inesistente.

Tentare di valutare allora le prestazioni del sistema distribuito così progettato equivale allora a valutare se l'accesso tramite il VL possa essere di effettiva utilità o se ad esempio i ritardi introdotti dalle comunicazioni possano rendere vano il progetto, ad esempio nel caso di un tentativo di Remote Monitoring tramite il protocollo Continuous Flow.

Dopo una prima fase di autenticazione, il *tempo di risposta dell'utente* per l'utilizzo del **protocollo Request/Reply** (ovvero il tempo che intercorre tra l'invio della sua Request e la ricezione della corrispondente Reply) è dato dalla somma seguente:

$$t = t_{andata} + t_{check} + t_{exec} + t_{ritorno}$$

dove t_{andata} e $t_{ritorno}$ sono i tempi richiesti dal messaggio per viaggiare nei due sensi sulla rete, t_{check} sono tutti i tempi impiegati dal Lab Server per decifrare e spacchettare il messaggio e per controllare che esso non violi nessuno dei requisiti di sicurezza richiesti, ed infine t_{exec} è il tempo effettivamente richiesto per la effettuazione del comando da parte del server e per la rilettura dei risultati.

Come si evince dalla formula, poiché il Lab Server svolge le proprie operazioni in maniera molto veloce (a parte la fase di autenticazione che in questo caso si è considerata già conclusa), ed il tempo di effettiva esecuzione non può essere modificato, tutto il tempo in più o in meno che può essere richiesto dalla effettuazione di un comando deve essere imputabile esclusivamente alle comunicazioni lungo la rete.

Da questo si ricava che, sebbene il sistema implementato sia il più generale possibile e possa essere utilizzato su Internet rendendo ad ogni utente possibile l'accesso a risorse situate fisicamente dalla parte opposta del mondo, solamente all'interno di una Intranet aziendale possono ottenersi dei risultati che soddisfino dei requisiti temporali strettamente stringenti.

In particolare utilizzando un **protocollo Continuous Flow** anziché uno Request/Reply l'utente si trova a guadagnare, per ogni risposta ricevuta, sia il t_{andata} che il t_{check}, per cui

una tale implementazione diventa più veloce proprio quando è richiesto il monitoraggio continuo di certe variabili di stato dell'impianto.

Per ognuno dei due protocolli, c'è inoltre da considerare che, adottando un opportuno algoritmo di *job allocation* su più computer slave, il t_{exec} può essere ulteriormente ridotto venendo ad ottenere prestazioni ancora migliori.

17 - Conclusioni

17.1 Aspetti innovativi

Dopo avere analizzato da tutti i punti di vista il progetto e la realizzazione del Laboratorio Virtuale è possibile dare di esso un giudizio qualitativo confrontando le possibilità che esso offre agli utenti con quelle che la tecnologia attuale permette di raggiungere.

In particolare ci si rifarà alle finalità che ci era prefissi nel primo capitolo, per verificare se siano state raggiunte e se abbiano effettivamente comportato una innovazione nel modo di lavorare e di apprendere degli utenti cui il VL si rivolge.

Un paragrafo a sé meritano i commenti circa la sicurezza delle risorse messe a disposizione, in quanto la utilità della realizzazione per gli utenti non deve entrare in conflitto con la necessità di salvaguardare ciò che viene reso accessibile da accessi indesiderati o da un utilizzo errato.

17.2 Apprendimento a distanza

Il Laboratorio Virtuale viene ad essere un ottimo strumento per la didattica; grazie al remote tutoring gli studenti vengono ad avere un facile accesso a tutorial e tool di simulazione e di valutazione automatica, il tutto comodamente dal proprio PC e con aggiornamenti in tempo reale.

Attualmente le uniche possibilità offerte al riguardo, al di fuori del VL, sono la consultazione di file HTML tramite il proprio browser; invece l'accesso remoto ai vari tool di simulazione è possibile solo mediante *telnet* su stazioni di lavoro che permettono di interagire con esse solo mediante un terminale virtuale a caratteri.

E' da osservare infatti che talune forme di collegamento remoto in modalità grafica presenti sui più recenti sistemi operativi (leggi Windows NT) possono aver luogo solamente all'interno di una rete locale, in quanto trasferire informazioni in formato grafico in tempo reale attraverso una rete geografica di computer è sicuramente impedito dalla esiguità della larghezza di banda disponibile.

La innovazione sostanziale offerta dal Laboratorio Virtuale viene invece ad essere la possibilità di utilizzare con il minimo sforzo, da qualunque punto della terra, una interfaccia in modalità grafica della complessità desiderata, in quanto come visto tale interfaccia verrà ad essere eseguita sul computer dell'utente, limitando lo scambio di informazioni remoto al minimo indispensabile.

L'utilizzo di una GUI di comprensione immediata elimina di fatto dal ciclo dell'apprendimento i tempi morti introdotti dalla dettagliata conoscenza del numero e del tipo di parametri che bisogna settare per ogni differente simulatore, nonché delle modalità di interpretazione dei dati prodotti in uscita da ognuno di essi.

Non ultimo, i vari tool da utilizzare possono essere virtualmente eseguiti su qualunque computer della terra, in maniera del tutto trasparente per gli utenti che si trovano a dover conoscere il solo indirizzo del VL, e sui differenti tipi di piattaforme hardware/software per le quali sono stati sviluppati.

Chiaramente il vantaggio della immediatezza dell'utilizzo dell'interfaccia grafica è tanto più grande quanto più è abile colui che si trova ad implementare la GUI; molto utile sarà ad

esempio una guida in linea che fornisca passo passo informazioni utili ad individuare le funzionalità offerte dal tool e le azioni compiute da ogni oggetto di controllo grafico.

Oltre alla GUI, innumerevoli sono le possibilità di intervento che uno studente potrebbe avere cimentandosi nell'interfacciamento con una API, potendo nel contempo mettere a frutto una mole di conoscenze circa i modelli Client/Server acquisita durante gli studi.

Grazie alla API è possibile sviluppare applicazioni personali che si interfaccino con le stesse risorse accessibili tramite le GUI; da notare che le stesse applicazioni personali possono prevedere interfacce grafiche della complessità desiderata, e qualora uno studente riesca a realizzare tramite la API una interfaccia con le stesse funzionalità e con una grafica più gradevole di quella offerta dalla GUI, proposta dall'applet accessibile tramite browser, non è difficile inglobare tale applicazione all'interno dell'applet stessa.

17.3 Controllo, monitoraggio a distanza e valutazione di prestazioni

In ambito industriale è molto sentita la necessità di un intervento rapido sulle varie parti di un impianto per monitorare l'andamento del ciclo produttivo o per intervenire in caso di malfunzionamenti del sistema.

La scelta che viene solitamente effetuata in ambito industriale è quella di non lasciare accessibile l'impianto dall'esterno, prevedendo solo una forma di controllo da parte di operatori specializzati su di appositi terminali; tale scelta viene solitamente effettuata solo per ovvi motivi di sicurezza, in quanto la maggior parte degli interventi prevede solo le interazioni tramite terminale senza che si renda necessaria effettivamente la presenza di tali operatori nei dintorni dell'impianto.

Qualora i requisiti di sicurezza offerti lo consentissero, la realizzazione di una forma di accesso remoto come quella proposta dal Laboratorio Virtuale consentirebbe alla impresa proprietaria dell'impianto di realizzare duplici vantaggi:

1. vantaggi economici, in quanto invece di tanti operatori dislocati per i vari impianti sparsi per il mondo sarebbero necessari solo pochi tecnici specializzati con capacità di intervento remoto su qualunque impianto;
2. vantaggi tecnici, in quanto potrebbero essere sempre i tecnici migliori e più preparati ad occuparsi di ogni impianti.

Il Laboratorio Virtuale permetterebbe allora di realizzare una forma di tele-lavoro, di cui tanto si parla oggi (sia pur per situazioni molto meno critiche, ovvero per il lavoro di ufficio) ma che quasi mai viene attuata più per motivi di consuetudine che per esigenze reali.

L'accesso remoto agli impianti per il controllo, il monitoraggio, la valutazione e la diagnosi dei malfunzionamenti è dunque una realtà, e per fare sì che i responsabili delle varie imprese si convincano dei vantaggi che una sua implementazione comporterebbe è necessario che le garanzie di sicurezza offerte siano soddisfacenti.

17.4 Sicurezza

Il Laboratorio Virtuale, così come è stato progettato, prevede diverse forme di controlli per garantire il massimo della sicurezza possibile sia per quanto riguarda gli accessi dalla rete che per quanto è inerente la sicurezza intrinseca degli impianti.

La sicurezza dagli accessi (esterni o interni) è stata perseguita con un opportuno protocollo di autenticazione, basato sugli algoritmi di cifratura DES e RSA che costituiscono

al momento lo stato dell’arte della crittografia, messo in atto tramite un complesso ed esauriente modello di sicurezza.

Lo stesso modello di sicurezza, oltre a prevedere flessibili forme di monitoraggio degli accessi, permette di modellare con qualunque grado di complessità le interazioni presenti all’interno degli impianti e le leggi che li regolano, giacché esse non dovranno essere catalogate rigidamente ma potranno essere cablate in software all’interno di una delle classi del Laboratorio Virtuale con il grado di dettaglio desiderato.

La serie di controlli compiuti dal modello di sicurezza è tale da garantire dunque anche la sicurezza intrinseca degli impianti, e così viene a cadere anche l’ultimo motivo per cui un tale Laboratorio Virtuale non potrebbe essere finalmente realizzato.

18 - Sorgenti della API del client

18.1 La classe LabClient

```
/*
 * @(#)LabClient.java
 *
 * Classe usata dall'applet per comunicare col LabServer
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/7
 */

import java.awt.*;
import java.net.*;
import java.io.*;

public class LabClient {
      // Classi di supporto private per gestione dati e cifratura
      private DataManager myDM;
      private CryptographyLibrary myCL;
      private long Session,Ticket;

      // Classi e variabili pubbliche
      public VirtualLab myBoss;
      public Socket sok;
      public byte buffer[] = new byte[1024];
      public int myPort;
```

```
	// Usato dal protocollo ContinuousFlow
	private String lastRequest[];

	// Costruttore apre il socket
	public LabClient(VirtualLab myIncomingBoss) {
		myBoss = myIncomingBoss;
		myDM = new DataManager();
		myCL = new CryptographyLibrary();

		System.out.println("Inizializzo connessione col server...");

		// Apre la connessione con il MultiLabServer sulla porta 6666
		try {
			sok=new Socket( InetAddress.getByName
				(myBoss.getCodeBase().getHost()),6666 );
		} catch (IOException e) { }

		if (sok!=null) {
			System.out.println("Connected to MultiLabServer:6666");
		} else {
			System.out.println("NOT connected to
MultiLabServer:6666");
		}

		// Riceve il numero di porta privato del SingleLabServer
		myPort = (int)myDM.twoBytesToLong(receiveBytes());
		try {
			sok=new Socket( InetAddress.getByName
				(myBoss.getCodeBase().getHost()), myPort );
		} catch (IOException e) { }

		if (sok!=null) {
```

```
                  System.out.println("Connected to
SingleLabServer:"+myPort);
            } else {
                  System.out.println("NOT connected to
SingleLabServer:"+myPort);
            }
      }

      // Invia String[] al server
      public void sendStrings(String data[]) {
            if (sok!=null) try {
                  OutputStream outstream;
                  outstream=sok.getOutputStream();
                  outstream.write(myDM.packStringsIntoBytes(data));
            } catch (IOException e) { }
      }

      // Invia String[] al server con cifratura DES
      public void sendStrings(String data[], long session, long ticket) {
            if (sok!=null) try {
                  OutputStream outstream;
                  outstream=sok.getOutputStream();
                  outstream.write( myCL.DESencryptCBC(
                        myDM.packStringsIntoBytes(data),
                        myDM.longToEightBytes(session),
                        myDM.longToEightBytes(ticket) ) );
            } catch (IOException e) { }
      }

      // Invia byte[] al server
      public void sendBytes(byte data[]) {
            if (sok!=null) try {
```

```
                OutputStream outstream;
                outstream=sok.getOutputStream();
                outstream.write(data);
        } catch (IOException e) { }
    }

    // Riceve String[] dal server
    public String[] receiveStrings() {
        String incomingData[] = null;

        if (sok!=null) try {
                InputStream instream;
                instream = sok.getInputStream();
                instream.read(buffer);
                incomingData = myDM.unpackStringsFromBytes(buffer);
        } catch (IOException e) { }

        return incomingData;
    }

    // Riceve String[] dal server con cifratura DES
    public String[] receiveStrings(long session, long ticket) {
        String incomingData[] = null;

        if (sok!=null) try {
                InputStream instream;
                instream = sok.getInputStream();
                instream.read(buffer);
                incomingData = myDM.unpackStringsFromBytes(
                        myCL.DESdecryptCBC(buffer,
                        myDM.longToEightBytes(session),
                        myDM.longToEightBytes(ticket) ) );
```

```
            } catch (IOException e) { }

            return incomingData;
      }

      // Riceve byte[] dal server
      public byte[] receiveBytes() {
            byte incomingData[] = null;
            int count;

            if (sok!=null) try {
                  InputStream instream;
                  instream = sok.getInputStream();
                  count = instream.read(buffer);
                  if (count!=-1) {
                        incomingData = new byte[count];
                        System.arraycopy(buffer,0,incomingData,0,count);
                  }
            } catch (IOException e) { }

            return incomingData;
      }

      // Chiude la connessione
      public void close() {
            if (sok!=null) try {
                  sok.close();
                  sok = null;
            } catch (IOException e) { }
      }

      // Gestisce il protocollo di autenticazione
```

```
      public boolean[] Authentication(String leavingData[]) {
            String inStrings[], outStrings[];
            byte inBytes[], outBytes[];
            boolean accessArray[] = null;

            if (sok != null) {
                  // Invia richiesta di connessione
                  outStrings = new String[1];
                  outStrings[0] = "connect";
                  sendStrings(outStrings);

                  // Riceve in chiaro chiave pubblica dell'algoritmo RSA
                  inBytes = receiveBytes();
                  byte temp[] = new byte[3];
                  System.arraycopy(inBytes,0,temp,0,3);
                  long n = myDM.threeBytesToLong(temp);
                  System.arraycopy(inBytes,3,temp,0,3);
                  long e = myDM.threeBytesToLong(temp);

                  // Genera, cifra RSA ed invia chiave di sessione
                  long session =
(long)(Math.random()*(double)Long.MAX_VALUE);
                  if (Math.random()>0.5) session = session*(-1);

      sendBytes(myCL.RSAencrypt(myDM.longToEightBytes(session),n,e));

                  // Da questo momento uso l'algoritmo DES con la chiave
                  // di sessione per tutte le comunicazioni

                  // Riceve il ticket di identificazione
                  long ticket = myDM.eightBytesToLong
                        ( myCL.DESencryptECB(receiveBytes(),
```

```
                        myDM.longToEightBytes(session)) );

                // Memorizza chiave e IV in variabili globali
                // (per farli usare poi ai protocolli)
                Session = session;
                Ticket = ticket;

                // spedisco userName e cryptedPassword
                sendStrings(leavingData,session,ticket);

                // ricevo i permessi
                inStrings = receiveStrings(session,ticket);

                // qui la stringa ricevuta va decifrata
                accessArray = myDM.stringToBooleans(inStrings[0]);
        }
        return accessArray;
    }

    // Gestisce il protocollo Request/Reply
    public String[] RequestReply(String request[]) {
        if (request == null) return null;

        String reply[] = null;

        if (sok != null) {
                // spedisco la richiesta
                sendStrings(request,Session,Ticket);

                // ricevo la risposta
                reply = receiveStrings(Session,Ticket);
        }
```

```
            return reply;
      }

      // Gestisce il protocollo Continuous Flow
      public String[] ContinuousFlow(String request[]) {
            String reply[] = null;
            boolean idem = true;

            if (request == null) return null;

            // Controllo se e' la stessa richiesta precedente
            if (request!=null && lastRequest!=null &&
                  request.length==lastRequest.length) {
                  for (int i=0; i<request.length; i++)
                        if (!request[i].equals(lastRequest[i])) idem =
false;

                  // E' la stessa richiesta precedente -> solo reply
                  if (idem) {
                        if (sok != null) {
                              // ricevo la risposta
                              reply = receiveStrings(Session,Ticket);
                        }
                  // Nuova richiesta
                  } else {
                        // Conservo richiesta attuale
                        lastRequest = new String[request.length];

      System.arraycopy(request,0,lastRequest,0,request.length);

                        if (sok != null) {
                              // spedisco la richiesta
```

```
                              sendStrings(request,Session,Ticket);

                              // ricevo la risposta
                              reply = receiveStrings(Session,Ticket);
                        }
                  }
            // Nuova richiesta
            } else {
                  // Conservo richiesta attuale
                  lastRequest = new String[request.length];
                  System.arraycopy(request,0,lastRequest,0,request.length);

                  if (sok != null) {
                        // spedisco la richiesta
                        sendStrings(request,Session,Ticket);

                        // ricevo la risposta
                        reply = receiveStrings(Session,Ticket);
                  }
            }
            return reply;
      }
}
```

18.2 La classe LabResources

```
/*
 * @(#)LabResources.java
 *
 * Contiene la descrizione delle risorse disponibili per il client
 *
 * @author  Fabrizio Fazzino
```

```
 * @version 1.0         1996/XI/15
 */

import java.io.*;

public class LabResources {
      // Variabili generali per memorizzare le risorse
      private static int numberOfResources = 4;
      private static final String resourceName[] = {
            "FabNet",
            "Resource2",
            "Resource3",
            "Resource4"
      };

      // Variabili usate dal client per ricordare i propri permessi
      private int numberOfAccessibleResources = 0;
      private boolean resourceAccess[];

      // Costruttore usato dal server non fa nulla
      public LabResources() {
      }

      // Costruttore (usato dal client) setta i permessi ricevuti.
      // Non importa se il client bara, tanto potrà solo ottenere
      // l'accesso all'interfaccia, mentre per le vere risorse il
      // server non guarda questa struttura ma utilizza le proprie
      public LabResources(boolean access[]) {
            if (access==null || access.length!=numberOfResources) {
                  System.err.println("Numero di permessi non valido");
                  System.exit(1);
            } else {
```

```
                  resourceAccess = access;
                  for (int i=0; i<resourceAccess.length; i++) {
                        if (resourceAccess[i])
numberOfAccessibleResources++;
                  }
            }
      }

      // Ritorna il numero di risorse totali
      public int getNumberOfResources() {
            return numberOfResources;
      }

      // Ritorna il numero di risorse accessibili
      public int getNumberOfAccessibleResources() {
            return numberOfAccessibleResources;
      }

      // Ritorna il permesso su ogni risorsa
      public boolean getResourceAccess(int n) {
            if (n>=0 && n<numberOfResources)
                  return resourceAccess[n];
            else return false;
      }

      // Restituisce dal numero il nome della risorsa
      public String getResourceName(int n) {
            if (n>=0 && n<numberOfResources)
                  return resourceName[n];
            else return null;
      }
```

```
	// Restituisce dal nome il numero della risorsa
	public int getResourceNumber(String name) {
		for (int i=0; i<numberOfResources; i++) {
			if (name.toLowerCase().startsWith
				(resourceName[i].toLowerCase()))
				return i;
		}
		return -1;
	}
}
```

19 - Sorgenti della GUI del client

19.1 Il file VirtualLab.html

```
<HTML>


<HEAD>
<TITLE>the Virtual Lab</TITLE>
</HEAD>


<BODY BGCOLOR="#FFFFFF" BACKGROUND="images/website.gif">


<CENTER>
<IMG SRC="images/vlabgra.gif"><IMG SRC="images/unictgra.gif">
<H1>Welcome to the Virtual Lab !</H1>
</CENTER>


<HR>


<H2><EM>Actual resources</EM></H2>
<P>Le risorse del Laboratorio Virtuale
attualmente disponibili sono le seguenti:
<DL>
<DT>Fab Net 3.0
<DD>Simulatore di reti locali di computer
con protocolli Token Bus, Token Ring e FDDI
<DT>Resource2
<DD>Attualmente non fa nulla
<DT>Resource3
```

```
<DD>Attualmente non fa nulla
<DT>Resource4
<DD>Attualmente non fa nulla
</DL>

<HR>

<APPLET
    code=VirtualLab.class
    id=VirtualLab
    width=320
    height=240 >
</APPLET>

<HR>

<A HREF="VirtualLab.java">Sorgente</a>

</BODY>

</HTML>
```

19.2 La applet VirtualLab

```
/*
 * @(#)VirtualLab.java
 *
 * Applet che implementa il lato client del Virtual Lab
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/X/29
 */
```

```
import java.applet.*;
import java.awt.*;

public class VirtualLab extends Applet implements Runnable {

      // Supporto per animazione richiede thread multipli
      Thread      myLabThread = null;                              // Thread
del laboratorio
      AnimationThread   myAnimationThread = null;     // Thread per
l'animazione

      // Variabili dell'AnimationThread
      public Graphics VL_Graphics;
      public Image VL_Images[];
      public int VL_CurrImageNo;
      public int VL_ImgWidth = 0;
      public int VL_ImgHeight = 0;
      public boolean VL_AllLoaded = false;
      public final int NUM_IMAGES = 18;

      // Bottone per il login
      private Button buttonLogin;

      // Variabili per sincronizzare i vari thread
      public boolean LoginDialogActive = false; // On=this, Off=dialog
      public boolean LoginOKPressed = false;          // On=dialog,
Off=this
      public String executionCommand = null;
      public boolean executionCanStart = false; // On=input, Off=this

      // Classi di supporto
```

```
	public LoginDialog myLoginDialog = null;
	public LabResources myResources = null;
	public LabClient myClient = null;

	public Frame myLoginDialogFrame = null;
	public LabResourcesFrame myResourcesFrame = null;
	public InputFrame myInputFrame = null;
	public OutputFrame myOutputFrame = null;

	// Variabili utilizzate per memorizzare info riservate
	public String userName = new String();
	private String cryptedPassword = new String();

	// Restituisce info dell'applet
	public String getAppletInfo() {
		return "Name: VirtualLab 1.0\r\n\n" +
			"Author: Fabrizio Fazzino\r\n" +
			"E-mail: ffazzino@k200.cdc.unict.it\r\n\n" +
			"Designed by the IIT of the University of Catania\r\n" +
			"From an idea of O.Mirabella & A.Di Stefano\r\n" +
			"With the help of L.Lo Bello\r\n\n" +
			"Release version January 1997";
	}

	// Inizializza applet aggiungendo bottone per login
	public void init() {
		buttonLogin = new Button("Login");
		add(buttonLogin);

		resize(320, 240);
	}
```

```
	// Controlla se viene premuto il bottone di login
	public boolean action(Event e, Object arg) {
		if (e.target instanceof Button && arg.equals("Login")) {
			if(!LoginDialogActive) {
				LoginDialogActive = true;

				myLoginDialogFrame = new Frame();
				myLoginDialog = new LoginDialog(this);
			}
		}
		return true;
	}

	// Avvia l'esecuzione dell'applet
	public void start() {
		if (myLabThread == null) {
			myLabThread = new Thread(this);
			myLabThread.setPriority(Thread.MAX_PRIORITY);
			myLabThread.start();
		}
		if (myAnimationThread == null) {
			myAnimationThread = new AnimationThread(this);
			myAnimationThread.setPriority(6);
			myAnimationThread.start();
		}
	}

	// Ferma l'esecuzione dell'applet
	public void stop() {
		if (myLabThread != null) {
			myLabThread.stop();
			myLabThread = null;
```

```
            }
            if (myAnimationThread != null) {
                  myAnimationThread.stop();
                  myAnimationThread = null;
            }
      }

      // Visualizza messaggio attesa caricamento oppure fotogramma
      public void paint(Graphics g) {
            if (VL_AllLoaded) {
                  Rectangle r = g.getClipRect();
                  g.clearRect(r.x, r.y, r.width, r.height);
                  displayImage(g);
            }
            else
                  g.drawString("Loading images...", 10, 20);
      }

      // Visualizza uno dei fotogrammi dell'animazione
      public void displayImage(Graphics g) {
            if (!VL_AllLoaded) return;

            g.drawImage(VL_Images[VL_CurrImageNo],
                        (size().width - VL_ImgWidth)   / 2,
                        (size().height - VL_ImgHeight) / 2, null);
      }

      // Aggiorna il numero dei fotogrammi caricati
      public boolean imageUpdate(Image img, int flags, int x, int y, int w,
int h) {
            if(VL_AllLoaded) return false;
```

```
            if ((flags & ALLBITS)==0) return true;

            if (++VL_CurrImageNo == NUM_IMAGES) {
                  VL_CurrImageNo = 0;
                  VL_AllLoaded = true;
            }

            return false;
      }

      // Gestisce le operazioni del VirtualLab sulla base
      // dello stato delle variabili di sincronizzazione
      public void run() {
            repaint();

            // Il ciclo seguente viene ripetuto all'infinito
            while (true) {
                  try {
                        // Leggo lo stato e svolgo tutte le operazioni

                        // Utente ha riempito il pannello di login
                        if(LoginOKPressed) {
                              LoginOKPressed = false;

                              // Inizializzo connessione ed autenticazione
                              myClient = new LabClient(this);
                              String data[] = new String[2];
                              data[0] = new String(userName);
                              data[1] = new String(cryptedPassword);
                              boolean access[] =
myClient.Authentication(data);
```

```
                              // Ricevo permessi e visualizzo risorse
                              if (access!=null && access.length>0) {
                                    myResources = new LabResources(access);
                                    myResourcesFrame = new
LabResourcesFrame(this);
                              } else {
                                    System.out.println("Accessi non
ricevuti");
                              }

                              // Può essere avviata la simulazione
                              while(true) {
                                    if(executionCanStart) {
                                          executionCanStart = false;

                                          // Viene avviato protocollo
Request/Reply
                                          String commands[] =
                                                { new
String(executionCommand) };
                                          String results[] =

      myClient.RequestReply(commands);
                                          if(results==null) {
                                                System.out.println("VL.run
fallito");
                                          } else {
                                                for (int zx=0;
zx<results.length; zx++) {

      System.out.println(results[zx]);
                                          }
```

```
      myOutputFrame.ShowInterface(results);
                                          }
                                    }
                              }
                        }
                  } catch (InterruptedException e) {
                        stop();
                  }
            }
      }

      // Usato per settare le variabili riservate
      public void setInfo(String un, String cp) {
            userName = new String(un);
            cryptedPassword = new String(cp);
      }
}
```

19.3 Il thread AnimationThread

```
/*
 * @(#)AnimationThread.java
 *
 * Thread separato che avvia l'animazione nell'applet
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/30
 */

import java.applet.*;
import java.awt.*;
```

```
public class AnimationThread extends Thread {
	VirtualLab myBoss = null;

	// Costruttore
	public AnimationThread(VirtualLab myIncomingBoss) {
		super();
		myBoss = myIncomingBoss;
	}

	// Carica i fotogrammi che compongono l'animazione
	public void run() {
		myBoss.repaint();

		myBoss.VL_Graphics = myBoss.getGraphics();
		myBoss.VL_CurrImageNo = 0;
		myBoss.VL_Images = new Image[myBoss.NUM_IMAGES];

		String strImage;

		// For each image in the animation, this method first
constructs a
		// string containing the path to the image file; then it begins
loading
		// the image into the VL_Images array.  Note that the call to
getImage
		// will return before the image is completely loaded.
		for (int i = 1; i <= myBoss.NUM_IMAGES; i++) {
			strImage = "images/img00" + ((i < 10) ? "0" : "") + i +
".gif";
		myBoss.VL_Images[i-1] =
myBoss.getImage(myBoss.getDocumentBase(),
```

```
                        strImage);

                if (myBoss.VL_ImgWidth == 0) {
                        try   {
                                // The getWidth() and getHeight() methods of
the Image class
                                // return -1 if the dimensions are not yet
known. The
                                // following code keeps calling getWidth()
and getHeight()
                                // until they return actual values (executed
only once).
                                while ((myBoss.VL_ImgWidth =
                                        myBoss.VL_Images[i-1].getWidth(null)) <
0)

                                        Thread.sleep(1);

                                while ((myBoss.VL_ImgHeight =
                                        myBoss.VL_Images[i-1].getHeight(null))
< 0)

                                        Thread.sleep(1);


                        } catch (InterruptedException e) { }
                }
                myBoss.VL_Graphics.drawImage(myBoss.VL_Images[i-1],
                        -1000,-1000,myBoss);
            }

            while (!myBoss.VL_AllLoaded) {
```

```
                  try   {
                        Thread.sleep(10);
                  } catch (InterruptedException e) { }
            }

            myBoss.repaint();

            // Il ciclo seguente viene ripetuto all'infinito
            while (true) {
                  try {
                        myBoss.displayImage(myBoss.VL_Graphics);
                        myBoss.VL_CurrImageNo++;
                        if (myBoss.VL_CurrImageNo == myBoss.NUM_IMAGES)
                              myBoss.VL_CurrImageNo = 0;

                        Thread.sleep(50);
                  } catch (InterruptedException e) {
                        stop();
                  }
            }
      }
}
```

19.4 La classe LoginDialog

```
/*
 * @(#) LoginDialog.java
 *
 * Finestra di dialogo per il login
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/X/31
```

```
 */

import java.awt.*;

public class LoginDialog extends Dialog {
      VirtualLab myBoss;

      // Variabili usate per il disegno
      public GridBagLayout myGrid = new GridBagLayout();
      public GridBagConstraints myGridPos = new GridBagConstraints();
      public Label UserNameLabel,PasswordLabel;
      public TextField UserNameField,PasswordField;
      public Button OKButton,CancelButton;

      // Usate per memorizzare le info da inviare all'applet
      private String userName,cryptedPassword;

      // Il costruttore disegna tutta la finestra di dialogo
      public LoginDialog (VirtualLab myIncomingBoss) {
            super (myIncomingBoss.myLoginDialogFrame, "Virtual Lab Login
Dialog", false);

            myBoss = myIncomingBoss;

            setLayout(myGrid);

            UserNameLabel = new Label("User Name : ");
            myGridPos.gridx=0;
            myGridPos.gridy=0;
            myGrid.setConstraints(UserNameLabel,myGridPos);
            add(UserNameLabel);
```

```
        PasswordLabel = new Label("Password : ");
        myGridPos.gridx=0;
        myGridPos.gridy=1;
        myGrid.setConstraints(PasswordLabel,myGridPos);
        add(PasswordLabel);

        UserNameField = new TextField(15);
        UserNameField.setEditable(true);
        myGridPos.gridx=1;
        myGridPos.gridy=0;
        myGrid.setConstraints(UserNameField,myGridPos);
        add(UserNameField);

        PasswordField = new TextField(15);
        PasswordField.setEchoCharacter('*');
        PasswordField.setEditable(true);
        myGridPos.gridx=1;
        myGridPos.gridy=1;
        myGrid.setConstraints(PasswordField,myGridPos);
        add(PasswordField);

        OKButton = new Button("OK");
        myGridPos.gridx=0;
        myGridPos.gridy=2;
        myGrid.setConstraints(OKButton,myGridPos);
        add(OKButton);

        CancelButton = new Button("Cancel");
        myGridPos.gridx=1;
        myGridPos.gridy=2;
        myGrid.setConstraints(CancelButton,myGridPos);
        add(CancelButton);
```

```
        reshape (200, 100, 250, 140);
        show();
        reshape (200, 100, 250, 140);
    }

    // Gestisce gli eventi che possono verificarsi
    public boolean handleEvent(Event ev) {
        // Chiudo la finestra di dialogo
        if (ev.id == Event.WINDOW_DESTROY) {
            myBoss.LoginDialogActive = false;
            dispose();
            return true;
        }
        // Premo uno dei bottoni
        if (ev.target instanceof Button) {
            String label = (String)ev.arg;
            if (label.equals("OK")) {

                userName = UserNameField.getText();
                cryptedPassword = PasswordField.getText();
                myBoss.setInfo(userName,cryptedPassword);
                myBoss.LoginDialogActive = false;
                myBoss.LoginOKPressed = true;

                dispose();
                return true;
            }
            if (label.equals("Cancel")) {
                myBoss.LoginDialogActive = false;
                dispose();
                return true;
```

```
                }
            }
            return false;
      }
}
```

19.5 La classe LabResourceFrame

```
/*
 * @(#)LabResourcesFrame.java
 *
 * Finestra per scegliere una delle risorse accessibili
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/19
 */

import java.awt.*;

public class LabResourcesFrame extends Frame {
      private VirtualLab myBoss;

      private GridBagLayout myGrid = new GridBagLayout();
      private GridBagConstraints myGridPos = new GridBagConstraints();
      private Label myLabel1 = new Label();
      private Label myLabel2 = new Label();
      private Label myLabel3 = new Label();
      private Button bottoniera[] = null;
      private Button cancelButton = null;

      // Costruttore visualizza le risorse disponibili
      public LabResourcesFrame(VirtualLab myIncomingBoss) {
```

```
            super ("Lab resources index");
            myBoss = myIncomingBoss;

            bottoniera = new
Button[myBoss.myResources.getNumberOfAccessibleResources()];

            setLayout(myGrid);

            // Stampa tre righe di testo (utente,porta,risorse)
            myLabel1.setText("You are the User : "+myBoss.userName);
            myGridPos.gridx = 0;
            myGridPos.gridy = 0;
            myGrid.setConstraints(myLabel1,myGridPos);
            add(myLabel1);

            myLabel2.setText("You are connected to LabServer port : "+
                  myBoss.myClient.myPort);
            myGridPos.gridx = 0;
            myGridPos.gridy = 1;
            myGrid.setConstraints(myLabel2,myGridPos);
            add(myLabel2);

            myLabel3.setText("You got authorized access to this resources :
");
            myGridPos.gridx = 0;
            myGridPos.gridy = 2;
            myGrid.setConstraints(myLabel3,myGridPos);
            add(myLabel3);

            // Visualizza un bottone per ogni risorsa autorizzata
            int buttonNumber = 0;
```

```
            for(int i=0; i<myBoss.myResources.getNumberOfResources(); i++)
{
                  if(myBoss.myResources.getResourceAccess(i)) {
                        bottoniera[buttonNumber++] = new
Button(myBoss.myResources.getResourceName(i));
                        myGridPos.gridx = 0;
                        myGridPos.gridy = buttonNumber+2;
                        myGrid.setConstraints(bottoniera[buttonNumber-
1],myGridPos);
                        add(bottoniera[buttonNumber-1]);
                  }
            }

            // Bottone per chiudere la finestra
            cancelButton = new Button("Cancel");
            myGridPos.gridx = 0;
            myGridPos.gridy = buttonNumber+5;
            myGrid.setConstraints(cancelButton,myGridPos);
            add(cancelButton);

            // Ridimensiona e ridisegna la finestra
            reshape(100,100,300,300);
            show();
            reshape(100,100,300,300);
      }

      // Gestisce la pressione dei bottoni
      public boolean handleEvent(Event ev) {
            // Chiudo la finestra di dialogo
            if (ev.id == Event.WINDOW_DESTROY) {
                  dispose();
                  return true;
```

```
            }
            // Premo uno dei bottoni
            if (ev.target instanceof Button) {
                  String label = (String)ev.arg;
                  if (label.equals("FabNet")) {
                        myBoss.myInputFrame = new
InputFrame(myBoss,"FabNet");
                        return true;
                  }
                  if (label.equals("Cancel")) {
                        dispose();
                        return true;
                  }
            }
            return false;
      }
}
```

19.6 La classe InputFrame

```
/*
 * @(#)InputFrame.java
 *
 * Finestra di ingresso alle risorse del laboratorio
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/8
 */

import java.awt.*;

public class InputFrame extends Frame {
```

```
      // Variabili generali
      private VirtualLab myBoss;
      private String resourceName;

      // Usate per formattare la disposizione dei componenti
      private GridBagLayout myGrid = new GridBagLayout();
      private GridBagConstraints myGridPos = new GridBagConstraints();

      // Pulsanti di avvio e di chiusura
      private Button OKbutton, cancelButton;

      // Componenti grafici (possono essere utilizzati da tutte le risorse)
      private Label myLabel0,myLabel1,myLabel2,myLabel3,myLabel4,myLabel5;
      private Label
myLabel6,myLabel7,myLabel8,myLabel9,myLabel10,myLabel11;
      private Button myButton0,myButton1,myButton2,myButton3,myButton4;
      private TextField myText0,myText1,myText2,myText3,myText4,myText5;
      private CheckboxGroup Radio1,Radio2,Radio3,Radio4,Radio5;
      private Checkbox Radio1A,Radio1B,Radio1C,Radio1D;
      private Checkbox Radio2A,Radio2B,Radio2C,Radio2D;
      private Checkbox Radio3A,Radio3B,Radio3C,Radio3D;
      private Checkbox Radio4A,Radio4B,Radio4C,Radio4D;
      private Checkbox Radio5A,Radio5B,Radio5C,Radio5D;

      // Costruttore setta il nome della risorsa cui si riferisce
      public InputFrame(VirtualLab myIncomingBoss, String
incomingResourceName) {
            super ("Input Interface of " + incomingResourceName);
            myBoss = myIncomingBoss;
            resourceName = incomingResourceName;

            ShowInterface();
```

```
}

// Visualizza una interfaccia differente per ogni risorsa
private void ShowInterface() {
      if(resourceName.equals("FabNet")) {
            setLayout(myGrid);

            // Titolo
            myLabel0 = new Label("FabNet 3.0");
            myGridPos.gridx = 1;
            myGridPos.gridy = 0;
            myGrid.setConstraints(myLabel0,myGridPos);
            add(myLabel0);

            // Protocollo
            myLabel1 = new Label("Protocol : ");
            myGridPos.gridx = 0;
            myGridPos.gridy = 1;
            myGrid.setConstraints(myLabel1,myGridPos);
            add(myLabel1);

            Radio1 = new CheckboxGroup();

            Radio1A = new Checkbox("Token Bus",Radio1,true);
            myGridPos.gridx = 1;
            myGridPos.gridy = 1;
            myGrid.setConstraints(Radio1A,myGridPos);
            add(Radio1A);

            Radio1B = new Checkbox("Token Ring",Radio1,false);
            myGridPos.gridx = 2;
            myGridPos.gridy = 1;
```

```
                myGrid.setConstraints(Radio1B,myGridPos);
                add(Radio1B);

                Radio1C = new Checkbox("FDDI",Radio1,false);
                myGridPos.gridx = 3;
                myGridPos.gridy = 1;
                myGrid.setConstraints(Radio1C,myGridPos);
                add(Radio1C);

                // Priorità
                myLabel2 = new Label("Priority : ");
                myGridPos.gridx = 0;
                myGridPos.gridy = 2;
                myGrid.setConstraints(myLabel2,myGridPos);
                add(myLabel2);

                Radio2 = new CheckboxGroup();

                Radio2A = new Checkbox("4",Radio2,true);
                myGridPos.gridx = 1;
                myGridPos.gridy = 2;
                myGrid.setConstraints(Radio2A,myGridPos);
                add(Radio2A);

                Radio2B = new Checkbox("3",Radio2,false);
                myGridPos.gridx = 2;
                myGridPos.gridy = 2;
                myGrid.setConstraints(Radio2B,myGridPos);
                add(Radio2B);

                Radio2C = new Checkbox("2",Radio2,false);
                myGridPos.gridx = 3;
```

```
                myGridPos.gridy = 2;
                myGrid.setConstraints(Radio2C,myGridPos);
                add(Radio2C);

                Radio2D = new Checkbox("1",Radio2,false);
                myGridPos.gridx = 4;
                myGridPos.gridy = 2;
                myGrid.setConstraints(Radio2D,myGridPos);
                add(Radio2D);

                // Nodi
                myLabel3 = new Label("Number of nodes : ");
                myGridPos.gridx = 0;
                myGridPos.gridy = 3;
                myGrid.setConstraints(myLabel3,myGridPos);
                add(myLabel3);

                myText0 = new TextField("30");
                myGridPos.gridx = 1;
                myGridPos.gridy = 3;
                myGrid.setConstraints(myText0,myGridPos);
                add(myText0);

                // Percorso
                myLabel4 = new Label("Path between two nodes [Km] : ");
                myGridPos.gridx = 0;
                myGridPos.gridy = 4;
                myGrid.setConstraints(myLabel4,myGridPos);
                add(myLabel4);

                myText1 = new TextField("1");
                myGridPos.gridx = 1;
```

```
                myGridPos.gridy = 4;
                myGrid.setConstraints(myText1,myGridPos);
                add(myText1);

                // Data Rate
                myLabel5 = new Label("Data Rate [Mbps] : ");
                myGridPos.gridx = 0;
                myGridPos.gridy = 5;
                myGrid.setConstraints(myLabel5,myGridPos);
                add(myLabel5);

                myText2 = new TextField("1");
                myGridPos.gridx = 1;
                myGridPos.gridy = 5;
                myGrid.setConstraints(myText2,myGridPos);
                add(myText2);

                // Lunghezza
                myLabel6 = new Label("Average message length [bit] : ");
                myGridPos.gridx = 0;
                myGridPos.gridy = 6;
                myGrid.setConstraints(myLabel6,myGridPos);
                add(myLabel6);

                myText3 = new TextField("1000");
                myGridPos.gridx = 1;
                myGridPos.gridy = 6;
                myGrid.setConstraints(myText3,myGridPos);
                add(myText3);

                // Tempo di simulazione
                myLabel7 = new Label("Time to simulate [msec] : ");
```

```
                myGridPos.gridx = 0;
                myGridPos.gridy = 7;
                myGrid.setConstraints(myLabel7,myGridPos);
                add(myLabel7);

                myText4 = new TextField("3000");
                myGridPos.gridx = 1;
                myGridPos.gridy = 7;
                myGrid.setConstraints(myText4,myGridPos);
                add(myText4);

                // Tempi di rotazione del token...

                reshape(10,10,600,400);
                show();
                reshape(10,10,600,400);
        }

        OKbutton = new Button("OK");
        myGridPos.gridx = 1;
        myGridPos.gridy = 8;
        myGrid.setConstraints(OKbutton,myGridPos);
        add(OKbutton);

        cancelButton = new Button("Cancel");
        myGridPos.gridx = 2;
        myGridPos.gridy = 8;
        myGrid.setConstraints(cancelButton,myGridPos);
        add(cancelButton);
    }

    // Gestisce la pressione dei tasti
```

```
      public boolean handleEvent(Event ev) {
            // Chiudo la finestra
            if (ev.id == Event.WINDOW_DESTROY) {
                  dispose();
                  return true;
            }
            // Premo uno dei bottoni
            if (ev.target instanceof Button) {
                  String label = (String)ev.arg;
                  if (label.equals("Cancel")) {
                        dispose();
                        return true;
                  }
                  if (label.equals("OK")) {
                        // Costruisce comando diverso a seconda della
risorsa

                        // Costruzione comando per 'FabNet'
                        if(resourceName.equals("FabNet")) {

                              StringBuffer command = new StringBuffer();

                              command.append("FABNET ");

                              if(Radio1A.getState()) command.append("0 ");
                              else if(Radio1B.getState()) command.append("1
");
                              else if(Radio1C.getState()) command.append("2
");
                              else command.append("* ");    // Murphy Law

                              if(Radio2A.getState()) command.append("4 ");
```

```
                                else if(Radio2B.getState()) command.append("3
");
                                else if(Radio2C.getState()) command.append("2
");
                                else if(Radio2D.getState()) command.append("1
");
                                else command.append("* ");    // Murphy Law

                                command.append(myText0.getText()+" ");
                                command.append(myText1.getText()+" ");
                                command.append(myText2.getText()+" ");
                                command.append(myText3.getText()+" ");
                                command.append(myText4.getText()+" ");

                                myBoss.executionCommand = new
String(command.toString());
                                myBoss.executionCanStart = true;
                           }
                           myBoss.myOutputFrame = new
OutputFrame(myBoss,resourceName);
                           return true;
                     }
               }
               return false;
         }
}
```

19.7 La classe OutputFrame

```
/*
 * @(#)OutputFrame.java
 *
```

```
 * Finestra per visualizzare i risultati delle interazioni
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/31
 */

import java.awt.*;

public class OutputFrame extends Frame {
      // Variabili generali
      private VirtualLab myBoss;
      private String resourceName;

      // Usate per formattare la disposizione dei componenti
      private GridBagLayout myGrid = new GridBagLayout();
      private GridBagConstraints myGridPos = new GridBagConstraints();

      // Separatori dei campi nelle stringhe
      private int recordSeparator = '#';
      private int fieldSeparator = ',';

      // Pulsanti di avvio e di chiusura
      private Button OKbutton, cancelButton;

      // Componenti grafici (possono essere utilizzati da tutte le risorse)
      private Image rect1,rect2,rect3,rect4,rect5,rect6;
      private float X1,X2,X3,X4,X5,X6;
      private float Y1,Y2,Y3,Y4,Y5,Y6;
      private Label myLabel0,myLabel1,myLabel2,myLabel3,myLabel4,myLabel5;
      private Label
myLabel6,myLabel7,myLabel8,myLabel9,myLabel10,myLabel11;
      private Button myButton0,myButton1,myButton2,myButton3,myButton4;
```

```
	private TextField myText0,myText1,myText2,myText3,myText4,myText5;
	private CheckboxGroup Radio1,Radio2,Radio3,Radio4,Radio5;
	private Checkbox Radio1A,Radio1B,Radio1C,Radio1D;
	private Checkbox Radio2A,Radio2B,Radio2C,Radio2D;
	private Checkbox Radio3A,Radio3B,Radio3C,Radio3D;
	private Checkbox Radio4A,Radio4B,Radio4C,Radio4D;
	private Checkbox Radio5A,Radio5B,Radio5C,Radio5D;

	// Costruttore setta il nome della risorsa cui si riferisce
	public OutputFrame(VirtualLab myIncomingBoss, String
incomingResourceName) {
		super ("Output Interface of " + incomingResourceName);
		myBoss = myIncomingBoss;
		resourceName = incomingResourceName;
	}

	// Visualizza una interfaccia differente per ogni risorsa
	public void ShowInterface(String results[]) {

		// Interfaccia di uscita del simulatore 'FabNet'
		if(resourceName.equals("FabNet")) {
			// Variabili di ingresso della simulazione
			// (nell'ordine: pro,pri,nod,pat,dat,len,tsi,tpr)
			int Protocol, Priority, Nodes, Length;
			float Path, DataRate, T_Simul;
			float T_Pri[] = new float[4];

			// Valori di ascisse e ordinate
			float WL_max, TP_max, AD_max;
			float WorkLoad[] = new float[8];

			// Risultati della simulazione
```

```
float ThroughPut[][] = new float[3][8];
float AccessDelay[][] = new float[3][8];

// Area di visualizzazione grafico ThroughPut
int TP_width = 210;
int TP_height = 140;

// Area di visualizzazione grafico Access Delay
int AD_width = 210;
int AD_height = 140;

// Rileggo parametri di ingresso
float floats[] = readFloatRecord(results[1]);
Protocol = (int)floats[0];
Priority = (int)floats[1];
Nodes = (int)floats[2];
Path = floats[3];
DataRate = floats[4];
Length = (int)floats[5];
T_Simul = floats[6];
for (int i=0; i<4; i++) T_Pri[i] = floats[7+i];

// Leggo i valori per normalizzare le ordinate
floats = readFloatRecord(results[2]);
TP_max = floats[0];
AD_max = floats[1];

// Leggo i valori delle ascisse
floats = readFloatRecord(results[3]);
for (int i=0; i<floats.length; i++)
      WorkLoad[i] = floats[i];
WL_max = WorkLoad[WorkLoad.length-1];
```

```
                // Trasmetto i valori max alle variabili globali (per
paint())
                X1 = WL_max;
                Y1 = TP_max;
                Y2 = AD_max;

                // Leggo risultati della simulazione
                int curves = ((Priority==4) ? 1 : 3 );
                int row=4;
                for (int a=0; a<curves; a++) {
                      floats = readFloatRecord(results[row+a]);
                      for (int i=0; i<floats.length; i++)
                            ThroughPut[a][i] = floats[i];
                }
                row = ( (Priority==4) ? 5 : 7 );
                for (int a=0; a<curves; a++) {
                      floats = readFloatRecord(results[row+a]);
                      for (int i=0; i<floats.length; i++)
                            AccessDelay[a][i] = floats[i];
                }

                // Volendo potrei ancora leggere dati per istogrammi...

                // Immagini per double-buffering e graphic context
                addNotify();      // crea il FramePeer (serve per
createImage)
                rect1 = createImage(TP_width, TP_height);
                rect2 = createImage(AD_width, AD_height);
                Graphics gc1 = rect1.getGraphics();
                Graphics gc2 = rect2.getGraphics();
```

```
                    // Sfondo grafico ThroughPut
                    gc1.setColor(Color.cyan);
                    gc1.fillRect(0,0,TP_width,TP_height);
                    gc1.setColor(Color.black);
                    // Sfondo grafico Access Delay
                    gc2.setColor(Color.pink);
                    gc2.fillRect(0,0,AD_width,AD_height);
                    gc2.setColor(Color.black);

                    // Inizio plottaggio grafici
                    float Xold,Yold,Xnew,Ynew;

                    // Curve ThroughPut (1 o 3)
                    for (int a=0; a<curves; a++) {
                          Xold = 0; Yold = 0;

                          for (int i=0; i<8; i++) {
                                Xnew = WorkLoad[i];
                                Ynew = Math.min(ThroughPut[a][i],TP_max);

                                gc1.drawLine(
                                      (int) (( Xold / WL_max )*
(float)TP_width),
                                      (int) (( 1.0 - Yold / TP_max )*
(float)TP_height),
                                      (int) (( Xnew / WL_max )*
(float)TP_width),
                                      (int) (( 1.0 - Ynew / TP_max )*
(float)TP_height ));

                                Xold = Xnew; Yold = Ynew;
                          }
```

```
                  }
                  // Curve Access Delay (1 o 3)
                  for (int a=0; a<curves; a++) {
                        Xold = 0; Yold = 0;

                        for (int i=0; i<8; i++) {
                              Xnew = WorkLoad[i];
                              Ynew = Math.min(AccessDelay[a][i],AD_max);

                              gc2.drawLine(
                                    (int) (( Xold / WL_max )*
(float)AD_width),
                                    (int) (( 1.0 - Yold / AD_max )*
(float)AD_height),
                                    (int) (( Xnew / WL_max )*
(float)AD_width),
                                    (int) (( 1.0 - Ynew / AD_max )*
(float)AD_height ));

                              Xold = Xnew; Yold = Ynew;
                        }
                  }
                  reshape(10,10,600,400);
                  show();
                  reshape(10,10,600,400);
            }
      }

      // Disegna la finestra
      public void paint(Graphics g) {
            if (resourceName.equals("FabNet") && rect1!=null &&
rect2!=null) {
```

```
                g.drawString("ThroughPut [Mbps] "+Y1,50,40);
                g.drawImage(rect1,250,40,this);
                g.drawString("Access Delay [msec] "+Y2,50,200);
                g.drawImage(rect2,250,200,this);
                g.drawString("WorkLoad [frames/sec] "+X1,250,350);
        }
    }

    // Gestisce la pressione dei tasti
    public boolean handleEvent(Event ev) {
        // Chiudo la finestra
        if (ev.id == Event.WINDOW_DESTROY) {
                dispose();
                return true;
        }
        return false;
    }

    // Modifica i separatori da utilizzare
    public void setSeparators(int rs, int fs) {
        recordSeparator = rs;
        fieldSeparator = fs;
    }

    // Converte una stringa (record) in array di interi
    public int[] readIntRecord(String record) {
        String fields[] = readRecord(record);
        int integers[] = new int[fields.length];

        for (int i=0; i<fields.length; i++) {
                try {
                        integers[i] = Integer.parseInt(fields[i]);
```

```
                } catch (NumberFormatException e) {
                        return null;
                }
        }

        return integers;
    }

    // Converte una stringa (record) in array di float
    public float[] readFloatRecord(String record) {
        String fields[] = readRecord(record);
        float floats[] = new float[fields.length];

        for (int i=0; i<fields.length; i++) {
                try {
                        floats[i] = Float.valueOf(fields[i]).floatValue();
                } catch (NumberFormatException e) {
                        return null;
                }
        }

        return floats;
    }

    // Converte una stringa (record) in array di stringhe
    public String[] readRecord(String record) {
        String strings[] = null;
        int numberOfFields = 1;
        int separatorPos = 0;
        int newPos = 0;

        // Conto il numero di campi
```

```
            while(true) {
                  newPos = record.indexOf(fieldSeparator,separatorPos);
                  if (newPos == -1) break;
                  else {
                        numberOfFields++;
                        separatorPos = newPos+1;
                  }
            }

            // Calcolo le posizioni di tutti i separatori
            int pos[] = new int[numberOfFields+1];
            pos[0] = record.indexOf(recordSeparator);
            if (pos[0] == -1) {
                  System.err.println("Record separator not found");
                  return null;
            }
            for (int k=1; k<numberOfFields; k++) {
                  pos[k] = record.indexOf(fieldSeparator,pos[k-1]+1);
            }
            pos[numberOfFields] = record.length();

            // Pongo i campi nelle stringhe da ritornare
            byte array[][] = new byte[numberOfFields][80];

            for (int x=0; x<numberOfFields; x++)
                  record.getBytes(pos[x]+1,pos[x+1],array[x],0);
            record = new String();

            strings = new String[numberOfFields];

            for (int x=0; x<numberOfFields; x++)
                  strings[x] = new String(new String(array[x],0)).trim();
```

```
            return strings;
      }
}
```

20 - Sorgenti del server

20.1 La applicazione LabServer

```
/*
 * @(#)LabServer.java
 *
 * Applicazione stand-alone che avvia il server del Virtual Lab
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/30
 */


public class LabServer {
      public static void main(String args[]) {
            new MultiLabServer().start();
      }
}
```

20.2 Il thread MultiLabServer

```
/*
 * @(#)MultiLabServer.java
 *
 * Istanzia un server singolo per ogni utente collegato
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/30
 */
```

```
import java.util.*;
import java.io.*;
import java.net.*;

public class MultiLabServer extends Thread {
	private DataManager myDM;

	private Socket sok = null;
	ServerSocket sersok1 = null;  // Uguale per tutti (richieste)

	// Costruttore attiva sersok1
	public MultiLabServer() {
		super("LabServer");
		try {
			sersok1 = new ServerSocket(6666);
			System.out.print("LabServer in ascolto sulla porta ");
			System.out.println(sersok1.getLocalPort());
		} catch (IOException e) {
			System.err.println("Impossibile attivare il LabServer");
		}
		myDM = new DataManager();
	}

	// Metodo run() del Thread attiva un sersok2 per ogni richiesta
	public void run() {
		try {
			if (sersok1!=null) {

				// Cicla all'infinito attendendo nuove connessioni
				while (true) {
					sok = sersok1.accept();
```

```
                                    ServerSocket sersok2 = new
ServerSocket(0,1);
                                    new SingleLabServer(sersok2).start();
                                    byte port[] = myDM.longToTwoBytes
                                          ((long)sersok2.getLocalPort());
                                    sendBytes(port);
                                    sok.close();
                                    sersok1.close();
                                    sersok1 = new ServerSocket(6666);
                                    System.out.print("LabServer in ascolto
sulla porta ");

      System.out.println(sersok1.getLocalPort());
                        }
                  } else {
                        sersok1 = new ServerSocket(6666);
                        System.out.print("LabServer in ascolto sulla porta
");
                        System.out.println(sersok1.getLocalPort());
                  }
            } catch (IOException e) { }
      }

      // Metodo finalize() del Thread (per garbage collection)
      protected void finalize() {
            if (sok!=null) try {
                  sok.close();
                  System.out.println("Chiudo il socket");
            } catch (IOException e) { }
      }

      // Invia byte[] al client
```

```
      public void sendBytes(byte data[]) {
            if (sok!=null) try {
                  OutputStream outstream;
                  outstream=sok.getOutputStream();
                  outstream.write(data);
            } catch (IOException e) { }
      }
}
```

20.3 Il thread SingleLabServer

```
/*
 * @(#)SingleLabServer.java
 *
 * Singolo thread del server per gestire i protocolli
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/9
 */

import java.net.*;
import java.io.*;

public class SingleLabServer extends Thread {
      // Classi di supporto per gestione dati e cifratura
      private DataManager     myDM;
      private CryptographyLibrary myCL;
      private LabSecurityManager myLSM;

      // Classi per la trasmissione sulla rete
      private static ServerSocket sersok;
      private static Socket sok;
```

```
	private static byte buffer[] = new byte[1024];

	// Variabili private
	private int port;
	private boolean protocol = false;   // solitamente Request/Reply
	private String userName, cryptedPassword;

	// Inizializza il server su una porta privata
	public SingleLabServer(ServerSocket mysersok) {
		super();

		sersok = mysersok;
		port = mysersok.getLocalPort();
		System.out.println("SingleLabServer in ascolto sulla porta
"+port);

		myDM = new DataManager();
		myCL = new CryptographyLibrary();
		myLSM = new LabSecurityManager();
	}

	// Metodo run() serve le richieste dei client
	public void run() {
		String inStrings[], outStrings[];
		byte inBytes[], outBytes[];

		try {
			try {
				// Accetta la richiesta di un client
				sok = sersok.accept();

				// Riceve richiesta di connessione in chiaro
```

```
                        inStrings = receiveStrings();
                        if (inStrings==null) close();
                        else if (inStrings.length==1 &&
                              inStrings[0].equals("connect")) {

                              // L'algoritmo RSA viene usato per lo scambio
della
                              // chiave di sessione

                              // Avvia RSA ed invia chiave pubblica (n,e)
in chiaro
                              long keys[] = myCL.RSAstart();
                              long n = keys[0];
                              long e = keys[1];
                              long d = keys[2];
                              outBytes = new byte[6];

      System.arraycopy(myDM.longToThreeBytes(n),0,outBytes,0,3);

      System.arraycopy(myDM.longToThreeBytes(e),0,outBytes,3,3);
                              sendBytes(outBytes);

                              // Riceve chiave di sessione e la decifra con
RSA
                              long session = myDM.eightBytesToLong
                                    (myCL.RSAdecrypt(receiveBytes(),n,d));

                              // Da questo momento tutte le comunicazioni
sono cifrate
                              // usando il DES e la chiave di sessione
```

```
                              // Genera ed invia il ticket da usare come
identificatore
                              long ticket =
(long)(Math.random()*(double)Long.MAX_VALUE);
                              if (Math.random()>0.5) ticket = ticket*(-1);

      sendBytes(myCL.DESencryptECB(myDM.longToEightBytes(ticket),
                                    myDM.longToEightBytes(session)));

                              // Riceve UserName e Password cifrate
                              inStrings = receiveStrings(session,ticket);
                              userName = new String(inStrings[0]);
                              cryptedPassword = new String(inStrings[1]);

                              // Autentica l'utente e manda autorizzazioni
                              String access =
myLSM.getAccesses(userName,cryptedPassword);
                              String accesses[] = { new String(access) };
                              sendStrings(accesses,session,ticket);

                              // Protocollo Request/Reply o Continuous Flow
                              while (true) {
                                    // Riceve ed esegue comando
                                    inStrings =
receiveStrings(session,ticket);

                                    if (
myLSM.checkAccess(userName,inStrings[0]) &&
                                          myLSM.checkCommand(inStrings) ) {

                                          System.out.println("Eseguo :
"+inStrings[0]);
```

```
                                        ExecutionThread myET = new
ExecutionThread(inStrings[0]);

                                        // Manda i risultati della
operazione remota
                                        outStrings =
myET.getExecutionResponse();

      sendStrings(outStrings,session,ticket);
                                  } else {
                                        System.out.println("Ricevuta
richiesta non sicura.");
                                        close();
                                  }
                            }
                      } else {
                            System.out.println("Connect non ricevuto");
                            close();
                      }
                } catch (IOException e) { }
          } catch (Exception n) { }
      }

      // Chiude la singola connessione
      private void close() throws IOException {
            if (sok!=null) {
                  sok.close();
                  sok = null;
            }
      }

      // Invia String[] al client
```

```
	public void sendStrings(String data[]) {
		if (sok!=null) try {
			OutputStream outstream;
			outstream=sok.getOutputStream();
			outstream.write(myDM.packStringsIntoBytes(data));
		} catch (IOException e) { }
	}

	// Invia String[] al client con cifratura DES
	public void sendStrings(String data[], long session, long ticket) {
		if (sok!=null) try {
			OutputStream outstream;
			outstream=sok.getOutputStream();
			outstream.write( myCL.DESencryptCBC(
				myDM.packStringsIntoBytes(data),
				myDM.longToEightBytes(session),
				myDM.longToEightBytes(ticket) ) );
		} catch (IOException e) { }
	}

	// Invia byte[] al client
	public void sendBytes(byte data[]) {
		if (sok!=null) try {
			OutputStream outstream;
			outstream=sok.getOutputStream();
			outstream.write(data);
		} catch (IOException e) { }
	}

	// Riceve String[] dal client
	public String[] receiveStrings() {
		String incomingData[] = null;
```

```
        if (sok!=null) try {
                InputStream instream;
                instream = sok.getInputStream();
                instream.read(buffer);
                incomingData = myDM.unpackStringsFromBytes(buffer);
        } catch (IOException e) { }

        return incomingData;
    }

    // Riceve String[] dal client con cifratura DES
    public String[] receiveStrings(long session, long ticket) {
        String incomingData[] = null;

        if (sok!=null) try {
                InputStream instream;
                instream = sok.getInputStream();
                instream.read(buffer);
                incomingData = myDM.unpackStringsFromBytes(
                        myCL.DESdecryptCBC(buffer,
                        myDM.longToEightBytes(session),
                        myDM.longToEightBytes(ticket) ) );
        } catch (IOException e) { }

        return incomingData;
    }

    // Riceve byte[] dal client
    public byte[] receiveBytes() {
        byte incomingData[] = null;
        int count;
```

```
            if (sok!=null) try {
                  InputStream instream;
                  instream = sok.getInputStream();
                  count = instream.read(buffer);
                  if (count!=-1) {
                        incomingData = new byte[count];
                        System.arraycopy(buffer,0,incomingData,0,count);
                  }
            } catch (IOException e) { }

            return incomingData;
      }
}
```

20.4 La classe LabSecurityManager

```
/*
 * @(#)LabSecurityManager.java
 *
 * Gestisce il Modello di Sicurezza del Virtual Lab
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1997/I/1
 */

class LabSecurityManager {
      // Variabili private
      private LabResources myLR;                // risorse del laboratorio
      private int numberOfResources;            // numero di risorse

      // Vettore dei bloccaggi -> Current Blocking Set
```

```
	private boolean Locks[];

	// Il costruttore inizializza il vettore dei bloccaggi
	public LabSecurityManager() {
		myLR = new LabResources();
		numberOfResources = myLR.getNumberOfResources();

		Locks = new boolean[numberOfResources];
		for (int i=0; i<Locks.length; i++) Locks[i] = false;
	}

	// Verifica che l'utente sia registrato e restituisce accessi
	// (spediti solo nella fase di autenticazione e senza alcun impegno)
	String getAccesses(String userName, String cryptedPassword) {
		String strings[] = null;
		String access = new String();

		FileManager myFile = new FileManager("data/access","r");

		while(true) {
			strings = myFile.readRecord();
			if (strings==null || strings.length<3) {
				break;
			} else if (strings[0].equals(userName) &&
strings[1].equals(cryptedPassword)) {
				access = new String(strings[2]);
				break;
			}
		}
		myFile.close();

		return access;
```

```
	}

	// Controlla nella Access Matrix (ovvero nel file 'data/access')
	// se l'utente autenticato ha il permesso di accedere alla
	// risorsa richiesta
	boolean checkAccess(String userName, String resourceName) {
		int num = myLR.getResourceNumber(resourceName);

		// Se la risorsa esiste
		if (num!=-1) {

			String records[];
			String accesses;

			FileManager myFile = new FileManager("data/access","r");

			while(true) {
				// Legge i permessi dalla Access Matrix
				records = myFile.readRecord();

				if (records==null || records.length<3) {
					break;
				} else if (records[0].equals(userName)) {
					accesses = new String(records[2]);

					// Restituisce se l'utente ha il permesso
					myFile.close();
					if (accesses.charAt(num)=='1')
						return true;
					else return false;
				}
			}
```

```
                  myFile.close();
            }
            return false;
      }

      // Controlla (in maniera anche attiva) se il comando da
      // eseguire possa essere pericoloso per gli impianti.
      // Implementa il Resources Interaction Set, e controlla
      // nel Current Blocking Set (vettore Locks[])
      boolean checkCommand(String commands[]) {
            // Il simulatore e' innocuo
            if (commands[0].toLowerCase().startsWith("fabnet")) {

                  return true;
            }

            return false;
      }
}
```

20.5 La classe FileManager

```
/*
 * @(#)FileManager.java
 *
 * Classe di supporto per accedere ai campi delle strutture dati
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/14
 */

import java.io.*;
```

```
public class FileManager {
      private RandomAccessFile myFile = null;
      private int FileIndex = 0;
      private int recordSeparator = '#';
      private int fieldSeparator = ',';

      // Costruttore apre il file da leggere o scrivere
    // 'name' e' il nome del file da aprire
    // 'mode' e' "r" per lettura o "rw" per lettura-scrittura
      public FileManager(String name, String mode) {
            try {
                  myFile = new RandomAccessFile(name, mode);
            } catch (IOException e) {
                  System.err.println("Problems accessing " + name);
                  System.exit(1);
            }
      }

      // Modifica i separatori da utilizzare
      public void setSeparators(int rs, int fs) {
            recordSeparator = rs;
            fieldSeparator = fs;
      }

      // Aggiunge un record al file
      public void addRecord(String strings[]) {
            try {
                  myFile.seek(myFile.length());
                  myFile.writeByte(recordSeparator);
                  for (int z=0; z<strings.length; z++) {
                        myFile.writeBytes(strings[z]);
```

```
                        if(z != (strings.length-1))
myFile.writeByte(fieldSeparator);
                  }
                  myFile.writeBytes("\r\n");
            } catch (IOException e) {
                  System.err.println("Problems adding record");
                  System.exit(1);
            }
      }

      // Legge un record dal file
      public String[] readRecord() {
            String strings[] = null;
            int numberOfFields = 1;
            int separatorPos = 0;
            int newPos = 0;
            String line = new String();

            try {
                  if (FileIndex < myFile.length()) {
                        // Leggo una riga
                        myFile.seek(FileIndex);
                        line = new String(myFile.readLine());
                        FileIndex += line.length()+1;

                        // Conto il numero di campi
                        while(true) {
                              newPos =
line.indexOf(fieldSeparator,separatorPos);
                              if (newPos == -1) break;
                              else {
                                    numberOfFields++;
```

```
                                    separatorPos = newPos+1;
                              }
                        }

                        // Calcolo le posizioni di tutti i separatori
                        int pos[] = new int[numberOfFields+1];
                        pos[0] = line.indexOf(recordSeparator);
                        if (pos[0] == -1) {
                              System.err.println("Record separator not
found");
                              System.exit(1);
                        }
                        for (int k=1; k<numberOfFields; k++) {
                              pos[k] = line.indexOf(fieldSeparator,pos[k-
1]+1);
                        }
                        pos[numberOfFields] = line.indexOf('\r');

                        // Pongo i campi nelle stringhe da ritornare
                        byte array[][] = new byte[numberOfFields][80];

                        for (int x=0; x<numberOfFields; x++)
                              line.getBytes(pos[x]+1,pos[x+1],array[x],0);
                        line = new String();

                        strings = new String[numberOfFields];

                        for (int x=0; x<numberOfFields; x++)
                              strings[x] = new String(new
String(array[x],0)).trim();
                  } else {
                        strings = null;
```

```
                  }
            } catch (IOException e) {
                  strings = null;
                  System.err.println("IOE catched");
            }
            return strings;
      }

      // Chiude il file
      public void close() {
            try {
                  myFile.close();
                  myFile = null;
            } catch (IOException e) { }
      }
}
```

20.6 La classe ExecutionThread

```
/*
 * @(#)ExecutionThread.java
 *
 * Usato dal server per eseguire i comandi a distanza
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/19
 */

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

public class ExecutionThread {
```

```
	String command = new String();

	// Costruttore memorizza il comando da eseguire
	public ExecutionThread(String cmd) {
		if (cmd!=null) command = new String(cmd);
	}

	// Esegue il comando e restituisce la risposta
	public String[] getExecutionResponse() {
		String response[] = null;

		try {
			System.out.println("Prima del processo");
			Process p = Runtime.getRuntime().exec(command);

			DataInputStream stream = new DataInputStream(new
BufferedInputStream(p.getInputStream()));
			String newString = null;
			while((newString=stream.readLine())!=null)
response=addString(response,newString);

			System.out.println("Dopo il processo");
			p.destroy();

		} catch (Exception e) {
			System.out.println("Exception : "+ e);
			e.printStackTrace();
		}

		return response;
	}
```

```
	// Usato per accrescere la risposta run-time
	private String[] addString(String oldStrings[], String stringToAdd) {
		String newStrings[];

		if (oldStrings!=null) {
			newStrings = new String[oldStrings.length+1];


	System.arraycopy(oldStrings,0,newStrings,0,oldStrings.length);
			newStrings[newStrings.length-1] = stringToAdd;
		} else {
			newStrings = new String[1];
			newStrings[0] = stringToAdd;
		}

		return newStrings;
	}
}
```

20.7 Il comando addUser

```
/*
 * @(#)addUser.java
 *
 * Comando stand-alone usato per aggiungere utenti al file 'access'
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/9
 */

import java.io.*;
```

```
public class addUser {
	// Variabili ausiliarie per scrivere su file un record
	private static final int NUMBER_OF_FIELDS = 3;
	private static String strings[] = new String[NUMBER_OF_FIELDS];

	// Variabili private
	private static int numberOfRights;

	// Legge una stringa dallo 'stdin'
	private static String readLine() throws IOException {
		byte line[] = new byte[80];
		int i=0, b=0;

		while( (i<80) && ((b=System.in.read())!='\n') ) {
			line[i++] = (byte)b;
		}

		return(new String(line,0,0,i));
	}

	// Aggiunge il nuovo utente al file 'access'
	public static void main(String args[]) throws IOException {

		LabResources myLR = new LabResources();
		numberOfRights = myLR.getNumberOfResources();

		if (args.length == 0) {
			System.err.println("Insert new User Name : ");
			strings[0] = readLine();

			System.err.println("Insert Password : ");
			strings[1] = readLine();
```

```
                System.err.print("Insert Rights (string of ");
                System.err.println(numberOfRights+" '0' and '1') : ");
                strings[2] = readLine();

            } else if (args.length == NUMBER_OF_FIELDS &&
                  args[2].trim().length()==numberOfRights) {

                  strings = args;
            } else {
                  System.out.print("Syntax: java addUser USERNAME PASSWORD
");
                  System.out.println("RIGHTS("+numberOfRights+")");
                  System.exit(1);
            }

            FileManager myFile = new FileManager("data/access","rw");
            myFile.addRecord(strings);
            myFile.close();
            myFile = null;

            System.out.println("User added to data/access");
      }
}
```

20.8 Il comando listUsers

```
/*
 * @(#)listUsers.java
 *
 * Comando stand-alone usato per leggere la Access Matrix
 *
```

```
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/12
 */

import java.io.*;

public class listUsers {

    // Legge utenti e permessi dal file 'access' (Access Matrix)
      public static void main(String args[]) throws IOException {
            String strings[];

            FileManager myFile = new FileManager("data/access","r");

            while(true) {
                  strings = myFile.readRecord();
                  if(strings==null) break;
                  System.out.print("User "+strings[0]);
                  System.out.println(" has rights "+strings[2]);
            }

            myFile.close();
      }
}
```

20.9 Il comando listResources

```
/*
 * @(#)listResources.java
 *
 * Comando stand-alone usato per elencare le risorse disponibili
 *
```

```
 * @author  Fabrizio Fazzino
 * @version 1.0         1997/I/6
 */


public class listResources {
      public static void main(String args[]) {
            LabResources myLR = new LabResources();
            int MAX = myLR.getNumberOfResources();

            System.out.println("There are "+MAX+" resources available.");

            for (int i=0; i<MAX; i++) {
                  System.out.print("Resource nr."+i+" is named ");
                  System.out.println(myLR.getResourceName(i));
            }
      }
}
```

21 - Sorgenti delle classi di supporto

21.1 La classe CryptographyLibrary

```
/*
 * @(#)CryptographyLibrary.java
 *
 * Libreria di algoritmi crittografici RSA e DES
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/30
 */


public class CryptographyLibrary {
      private RSA myRSA;
      private DES myDES;

      // Costruttore
      public CryptographyLibrary() {
            myRSA = new RSA();
            myDES = new DES();
      }

      // Metodi dell'algoritmo RSA

      // Inizializzazione dell'algoritmo RSA
      // (torna long { n, e, d } )
      public long[] RSAstart() {
        return myRSA.RSAstart();
```

```
    }

    // Cifratura RSA byte[] -> byte[] ( dim. *2 -> *3 )
    // (ingresso e' riempito fino ad essere di lunghezza multipla di 2)
    // (l'uscita ha dimensione sempre multipla di 3)
    public byte[] RSAencrypt (byte plain[], long n, long e) {
      return myRSA.RSAencrypt(plain,n,e);
    }

    // Decifratura RSA byte[] -> byte[] ( dim. *3 -> *2 )
    public byte[] RSAdecrypt (byte cipher[], long n, long d) {
      return myRSA.RSAdecrypt(cipher,n,d);
    }

    // Metodi dell'algoritmo DES

    // Cifra l'array di byte in ingresso con il DES
    // secondo il Modo di Operazione ECB (Electronic Codebook)
    // ( byte[n*8] -> byte[n*8] )
    // ( se ingresso non e' [n*8] appendo padding casuali )
    public byte[] DESencryptECB(byte plaintext[], byte key[]) {
          return myDES.DESencryptECB(plaintext,key);
    }

    // Decifra array cifrato col DES in modo ECB
    public byte[] DESdecryptECB(byte ciphertext[], byte key[]) {
          return myDES.DESdecryptECB(ciphertext,key);
    }

    // Cifra l'array di byte in ingresso con il DES
    // secondo il Modo di Operazione CBC (Cipher Block Chaining)
    public byte[] DESencryptCBC(byte plaintext[], byte key[],
```

```
            byte IV[]) {

            return myDES.DESencryptCBC(plaintext,key,IV);
      }

      // Decifra l'array precedentemente cifrato in modo CBC
      public byte[] DESdecryptCBC(byte ciphertext[], byte key[],
            byte IV[]) {

            return myDES.DESdecryptCBC(ciphertext,key,IV);
      }
}
```

21.2 La classe DES

```
/*
 * @(#)DES.java
 *
 * Classe con tutti gli strumenti della cifratura DES
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/10
 */

public class DES {
      DataManager myDM; // alloca memoria per nuove strutture

      // Dati dell'algoritmo di cifratura DES

      // Vettori di trasposizione del testo (64 int [1..64])

      // IP
```

```
	private final int InitialTr[] = {
		58,50,42,34,26,18,10, 2,60,52,44,36,28,20,12, 4,
		62,54,46,38,30,22,14, 6,64,56,48,40,32,24,16, 8,
		57,49,41,33,25,17, 9, 1,59,51,43,35,27,19,11, 3,
		61,53,45,37,29,21,13, 5,63,55,47,39,31,23,15, 7 };

	// FP
	private final int FinalTr[] = {
		40, 8,48,16,56,24,64,32,39, 7,47,15,55,23,63,31,
		38, 6,46,14,54,22,62,30,37, 5,45,13,53,21,61,29,
		36, 4,44,12,52,20,60,28,35, 3,43,11,51,19,59,27,
		34, 2,42,10,50,18,58,26,33, 1,41, 9,49,17,57,25 };

	private final int swap[] = {
		33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,
		49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,
		 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,
		17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32 };

	// Vettori di trasposizione delle chiavi

	// PC-1 (56 int [1..64])
	private final int KeyTr1[] = {
		57,49,41,33,25,17, 9, 1,58,50,42,34,26,18,
		10, 2,59,51,43,35,27,19,11, 3,60,52,44,36,
		63,55,47,39,31,23,15, 7,62,54,46,38,30,22,
		14, 6,61,53,45,37,29,21,13, 5,28,20,12, 4 };

	// PC-2 (48 int [1..56])
	private final int KeyTr2[] = {
		14,17,11,24, 1, 5, 3,28,15, 6,21,10,
		23,19,12, 4,26, 8,16, 7,27,20,13, 2,
```

```
            41,52,31,37,47,55,30,40,51,45,33,48,
            44,49,39,56,34,53,46,42,50,36,29,32 };

      // Altri vettori

      // E (48 int [1..32])
      private final int etr[] = {
            32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,
             8, 9,10,11,12,13,12,13,14,15,16,17,
            16,17,18,19,20,21,20,21,22,23,24,25,
            24,25,26,27,28,29,28,29,30,31,32, 1 };

      // P (32 int [1..32])
      private final int ptr[] = {
            16, 7,20,21,29,12,28,17, 1,15,23,26, 5,18,31,10,
             2, 8,24,14,32,27, 3, 9,19,13,30, 6,22,11, 4,25 };

      // Rots (16 int [1;2])
      private final int rots[] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

      // S-boxes (blocchi di sostituzione) (matrice di 8*64 int):
      private final int s[][] = {
            {
                  14, 4,13, 1, 2,15,11, 8, 3,10, 6,12, 5, 9, 0, 7,
                   0,15, 7, 4,14, 2,13, 1,10, 6,12,11, 9, 5, 3, 8,
                   4, 1,14, 8,13, 6, 2,11,15,12, 9, 7, 3,10, 5, 0,
                  15,12, 8, 2, 4, 9, 1, 7, 5,11, 3,14,10, 0, 6,13 },
            {
                  15, 1, 8,14, 6,11, 3, 4, 9, 7, 2,13,12, 0, 5,10,
                   3,13, 4, 7,15, 2, 8,14,12, 0, 1,10, 6, 9,11, 5,
                   0,14, 7,11,10, 4,13, 1, 5, 8,12, 6, 9, 3, 2,15,
                  13, 8,10, 1, 3,15, 4, 2,11, 6, 7,12, 0, 5,14, 9 },
```

```
        {
              10, 0, 9,14, 6, 3,15, 5, 1,13,12, 7,11, 4, 2, 8,
              13, 7, 0, 9, 3, 4, 6,10, 2, 8, 5,14,12,11,15, 1,
              13, 6, 4, 9, 8,15, 3, 0,11, 1, 2,12, 5,10,14, 7,
               1,10,13, 0, 6, 9, 8, 7, 4,15,14, 3,11, 5, 2,12 },
        {
               7,13,14, 3, 0, 6, 9,10, 1, 2, 8, 5,11,12, 4,15,
              13, 8,11, 5, 6,15, 0, 3, 4, 7, 2,12, 1,10,14, 9,
              10, 6, 9, 0,12,11, 7,13,15, 1, 3,14, 5, 2, 8, 4,
               3,15, 0, 6,10, 1,13, 8, 9, 4, 5,11,12, 7, 2,14 },
        {
               2,14, 4, 1, 7,10,11, 6, 8, 5, 3,15,13, 0,14, 9,
              14,11, 2,12, 4, 7,13, 1, 5, 0,15,10, 3, 9, 8, 6,
               4, 2, 1,11,10,13, 7, 8,15, 9,12, 5, 6, 3, 0,14,
              11, 8,12, 7, 1,14, 2,13, 6,15, 0, 9,10, 4, 5, 3 },
        {
              12, 1,10,15, 9, 2, 6, 8, 0,13, 3, 4,14, 7, 5,11,
              10,15, 4, 2, 7,12, 9, 5, 6, 1,13,14, 0,11, 3, 8,
               9,14,15, 5, 2, 8,12, 3, 7, 0, 4,10, 1,13,11, 6,
               4, 3, 2,12, 9, 5,15,10,11,14, 1, 7, 6, 0, 8,13 },
        {
               4,11, 2,14,15, 0, 8,13, 3,12, 9, 7, 5,10, 6, 1,
              13, 0,11, 7, 4, 9, 1,10,14, 3, 5,12, 2,15, 8, 6,
               1, 4,11,13,12, 3, 7,14,10,15, 6, 8, 0, 5, 9, 2,
               6,11,13, 8, 1, 4,10, 7, 9, 5, 0,15,14, 2, 3,12 },
        {
              13, 2, 8, 4, 6,15,11, 1,10, 9, 3,14, 5, 0,12, 7,
               1,15,13, 8,10, 3, 7, 4,12, 5, 6,11, 0,14, 9, 2,
               7,11, 4, 1, 9,12,14, 2, 0, 6,10,13,15, 3, 5, 8,
               2, 1,14, 7, 4,10, 8,13,15,12, 9, 0, 3, 5, 6,11 }
    };
```

```
  // Costruttore
public DES() {
        myDM = new DataManager();           // alloca memoria
  }

  // Procedure interne all'algoritmo di cifratura DES

  // Traspone i primi n elementi di data con trasposizione trasp
  // Modifica: data[]
  // Invariati: t[], n
private void traspone(boolean data[], int trasp[], int n) {
    boolean olddata[] = new boolean[data.length];
    System.arraycopy(data,0,olddata,0,data.length);

    for (int i=0; i<n; i++) data[i] = olddata[trasp[i]-1];
  }

// Ruota key a sinistra di 1 bit su due unita' di 28 bit n volte
// Modifica: key[]
private void ruota(boolean key[], int n) {
    boolean oldkey[] = new boolean[key.length];

        for (int z=0; z<n; z++) {
        System.arraycopy(key,0,oldkey,0,key.length);
        for (int i=0; i<55; i++) key[i] = oldkey[i+1];
        key[27] = oldkey[0];
        key[55] = oldkey[28];
        }
  }

  // Funzione principale della cifratura
// Modifica: key[56], a[64](dove?), x[64]
```

```
      // Invariato: i
    private void f(int i, boolean key[], boolean a[], boolean x[]) {
            boolean e[] = new boolean[64];
        boolean ikey[] = new boolean[56];
            boolean y[] = new boolean[48];

        System.arraycopy(a,0,e,0,64);           // Copia testo cifrato in
'e'
        traspone(e,etr,48);                     // Espande primi 32->48 bit

        ruota(key,rots[i]);                     // Ruota la chiave

        System.arraycopy(key,0,ikey,0,key.length);  // Copia la chiave
        traspone(ikey,KeyTr2,48);               // Traspone chiave attuale

        for (int j=0; j<48; j++) y[j] = e[j]^ikey[j]; // y[48]=XOR con
chiave

            for (int k=1; k<=8; k++) {
            int r = (                           // 'r' ha valori in [0..63]
                        32*(y[6*k-6] ? 1 : 0) +
                        16*(y[6*k-1] ? 1 : 0) +
                         8*(y[6*k-5] ? 1 : 0) +
                         4*(y[6*k-4] ? 1 : 0) +
                         2*(y[6*k-3] ? 1 : 0) +
                         1*(y[6*k-2] ? 1 : 0)       );

            x[4*k-4] = myDM.dispari(s[k-1][r]/8);    // Riempio x[0..31]
                  x[4*k-3] = myDM.dispari(s[k-1][r]/4);
                  x[4*k-2] = myDM.dispari(s[k-1][r]/2);
                  x[4*k-1] = myDM.dispari(s[k-1][r]);
            }
```

```
        traspone(x, ptr, 32);                   // Mescola primi 32 di 'x'
      }

    // Algoritmo di cifratura DES privato (singolo blocco):
    // dati plaintext[64] e key[64] riempie ciphertext[64]
    // Modifica: ciphertext[] (distrugge anche key[])
    private boolean[] DEScrypt(boolean plaintext[], boolean longkey[]) {
            boolean ciphertext[] = new boolean[64];

            boolean a[] = new boolean[64];
            boolean b[] = new boolean[64];
            boolean x[] = new boolean[64];
        boolean shortkey[] = new boolean[56];

        System.arraycopy(plaintext,0,a,0,64);   // 'a' testo in chiaro
        traspone(a,InitialTr,64);               // Trasposizione iniziale

        traspone(longkey,KeyTr1,56);            // Riduce chiave a 56 bit
        System.arraycopy(longkey,0,shortkey,0,56);

        for (int i=0; i<16; i++) {              // 16 iterazioni
            System.arraycopy(a,0,b,0,64);       // 'a' testo cifrato
corrente
            for (int j=0; j<32; j++) a[j]=b[j+32];      // sx=dx

            f(i,shortkey,a,x);                          // x=f(r[i-1],k[i])

            for (int j=0; j<32; j++) a[j+32] = b[j]^x[j];   // dx=XOR
            }

        traspone(a,swap,64);                    // Scambia meta' sx/dx
        traspone(a,FinalTr,64);                 // Trasposizione finale
```

```
  System.arraycopy(a,0,ciphertext,0,64);  // Testo cifrato

      return ciphertext;
}

// Espande un array di byte fino ad avere una dimensione multipla
// di 8, appendendo dei caratteri di riempimento (padding) casuali
// (lo rende idoneo alla cifratura DES di blocchi di 64 bit)
private byte[] ExpandToBlock(byte input[]) {
      // Se e' gia' multiplo di un blocco non fa nulla
      if (input.length%8==0) return input;

      // Calcola dim attuale e dim richiesta
      int olddim = input.length;
      int newdim = ((int)(olddim/8)+1)*8;

      // Crea array piu' grande e ricopia i primi elementi
      byte output[] = new byte[newdim];
      System.arraycopy(input,0,output,0,olddim);

      // Riempie ultimi elementi con numeri casuali
      int count = newdim-olddim;
      while(count>0) {
            output[olddim+(--count)] =
                  (byte)((256*Math.random())%256-128);
      }
      return output;
}

// Cifra l'array di byte in ingresso con il DES
// secondo il Modo di Operazione ECB (Electronic Codebook)
```

```
public byte[] DESencryptECB(byte plaintext[], byte key[]) {

      if (plaintext==null || plaintext.length==0) return null;
      else plaintext = ExpandToBlock(plaintext);

      byte ciphertext[] = new byte[plaintext.length];

      int numofblocks = plaintext.length/8;

      boolean boolplain[] = new boolean[64];
      boolean boolkey[] = new boolean[64];
      boolean boolcipher[] = new boolean[64];

      boolkey = myDM.bytesToBooleans(key);

      byte tempblock[] = new byte[8];

      for (int zx=0; zx<numofblocks; zx++) {
            System.arraycopy(plaintext,zx*8,tempblock,0,8);
            boolplain = myDM.bytesToBooleans(tempblock);

            boolcipher = DEScrypt(boolplain, boolkey);

            System.arraycopy(myDM.booleansToBytes(boolcipher),0,
                  ciphertext,zx*8,8);
      }

      return ciphertext;
}

// Decifra l'array precedentemente cifrato in modo ECB
public byte[] DESdecryptECB(byte ciphertext[], byte key[]) {
```

```
        if (ciphertext==null || ciphertext.length==0 ||
              ((ciphertext.length%8)!=0) ) return null;

        byte plaintext[] = new byte[ciphertext.length];

        int numofblocks = ciphertext.length/8;

        boolean boolcipher[] = new boolean[64];
        boolean boolkey[] = new boolean[64];
        boolean boolplain[] = new boolean[64];

        boolkey = myDM.bytesToBooleans(key);

        byte tempblock[] = new byte[8];

        for (int zx=0; zx<numofblocks; zx++) {
              System.arraycopy(ciphertext,zx*8,tempblock,0,8);
              boolcipher = myDM.bytesToBooleans(tempblock);

              boolplain = DEScrypt(boolcipher, boolkey);

              System.arraycopy(myDM.booleansToBytes(boolplain),0,
                    plaintext,zx*8,8);
        }

        return plaintext;
    }

    // Per il modo di operazione CBC bisogna anche eseguire
    // lo XOR di blocchi di dati di 64 bit
    private boolean[] XOR(boolean a[], boolean b[]) {
```

```
        if (a.length!=b.length) return null;
        boolean c[] = new boolean[a.length];

        for (int i=0; i<c.length; i++) c[i]=a[i]^b[i];

        return c;
    }

    // Cifra l'array di byte in ingresso con il DES
    // secondo il Modo di Operazione CBC (Cipher Block Chaining)
    public byte[] DESencryptCBC(byte plaintext[], byte key[],
        byte IV[]) {

        if (plaintext==null || plaintext.length==0) return null;
        else plaintext = ExpandToBlock(plaintext);

        byte ciphertext[] = new byte[plaintext.length];

        int numofblocks = plaintext.length/8;

        boolean boolplain[] = new boolean[64];
        boolean boolkey[] = new boolean[64];
        boolean boolcipher[] = new boolean[64];

        boolkey = myDM.bytesToBooleans(key);
        boolcipher = myDM.bytesToBooleans(IV);

        byte tempblock[] = new byte[8];

        // Cifra un blocco alla volta
        for (int zx=0; zx<numofblocks; zx++) {
            System.arraycopy(plaintext,zx*8,tempblock,0,8);
```

```
                boolplain = myDM.bytesToBooleans(tempblock);

                boolplain = XOR(boolplain,boolcipher);

                boolcipher = DEScrypt(boolplain, boolkey);
                System.arraycopy(myDM.booleansToBytes(boolcipher),0,
                      ciphertext,zx*8,8);
          }

          return ciphertext;
    }

    // Decifra l'array precedentemente cifrato in modo CBC
    public byte[] DESdecryptCBC(byte ciphertext[], byte key[],
          byte IV[]) {

          if (ciphertext==null || ciphertext.length==0) return null;
          else ciphertext = ExpandToBlock(ciphertext);

          byte plaintext[] = new byte[ciphertext.length];

          int numofblocks = ciphertext.length/8;

          boolean boolcipher[] = new boolean[64];
          boolean boolkey[] = new boolean[64];
          boolean boolplain[] = new boolean[64];
          boolean boolmem[] = new boolean[64];

          boolkey = myDM.bytesToBooleans(key);
          boolmem = myDM.bytesToBooleans(IV);

          byte tempblock[] = new byte[8];
```

```
            // Decifra un blocco alla volta
            for (int zx=0; zx<numofblocks; zx++) {
                  System.arraycopy(ciphertext,zx*8,tempblock,0,8);
                  boolcipher = myDM.bytesToBooleans(tempblock);

                  boolplain = DEScrypt(boolcipher, boolkey);

                  boolplain = XOR(boolplain,boolmem);
                  System.arraycopy(boolcipher,0,boolmem,0,64);

                  System.arraycopy(myDM.booleansToBytes(boolplain),0,
                        plaintext,zx*8,8);
            }

            return plaintext;
      }
}
```

21.3 La classe RSA

```
/*
 * @(#)RSA.java
 *
 * Classe con tutti gli strumenti per la cifratura RSA
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XII/27
 */

public class RSA {
      DataManager myDM;
```

```
// Costruttore
public RSA() {
	myDM = new DataManager();
}

// Algoritmo di Euclide per trovare il Massimo Comune Divisore
// (gcd = greatest common divisor) di due numeri interi
public long Euclid(long d, long f) {
	long X = f;
	long Y = d;
	long R;

	while (true) {
		if (Y==0) return X;
		R = X % Y;
		X = Y;
		Y = R;
	}
}

// Algoritmo Esteso di Euclide per trovare il MCD (in long[0])
// e se possibile (MCD=1) anche l'inverso moltiplicativo (long[1])
// ( se l'inverso non esiste torna long[1]=0 )
public long[] ExtendedEuclid(long d, long f) {
	long out[] = { 0, 0 };
	long X1,X2,X3,Y1,Y2,Y3,T1,T2,T3,Q;

	X1=1; X2=0; X3=f; Y1=0; Y2=1; Y3=d;

	while (true) {
		if (Y3==0) {
```

```
                        out[0] = X3;            // MCD
                        return out;             // torna { MCD, 0 }
                  }
                  if (Y3==1) {
                        out[0] = Y3;            // MCD
                        if (Y2>0 && (((d*Y2)%f)==1))
                              out[1] = Y2;      // inverso (inv d mod f)
                        else
                              out[1] = 0;       // per la Legge di Murphy
                        return out;             // torna { MCD=1, inverso }
                  }
                  Q = X3 / Y3;            // prendo il quoziente
                  T1=X1-Q*Y1; T2=X2-Q*Y2; T3=X3-Q*Y3;
                  X1=Y1; X2=Y2; X3=Y3;
                  Y1=T1; Y2=T2; Y3=T3;
            }
      }

      // Il test di primalita' di Rabin Miller dice se e' probabile che
      // l'argomento sia un numero primo (esegue il test singolo 5 volte)
      public boolean RabinMiller(long p) {
            return RabinMiller(p, 5);
      }

      // Il test di primalita' di Rabin Miller dice se e' probabile che
      // l'argomento sia un numero primo (esegue test singolo 'times'
volte)
      public boolean RabinMiller(long p, int times) {
            long test[] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47};
            long a,b,m;

            // Elimino le estrazioni errate
```

```
            if (p<=1) return false;

            // Se e' divisibile per uno dei numeri di test non e' primo
            for (int x=0; x<test.length; x++) {
                  if ((p%test[x])==0) return false;
            }

            // 'b' e' l'exp della max pot di 2 che divide p-1
            long temp = p-1;
            b = 0;
            while(true) {
                  if (!myDM.dispari(temp)) {
                        b++;
                        temp = temp/2;
                  } else {
                        break;
                  }
            }

            // 'm' e' tale che p=1+pow(2,b)*m
            m = (long)((p-1)/(long)Math.pow(2,b));

            // E' probabilmente primo se supera il test 'times' volte
            for (int i=0; i<times; i++) {

                  // Genera un numero casuale a<p
                  do {
                        a = (long)(Math.random()*(double)p);
                  } while(a>=p || a==0);

                  if(!singleRabinMiller(p,a,b,m)) return false;
            }
```

```
            return true;
      }

      // Procedura usata per un singolo test di primalita'
      // (la procedura pubblica chiama questa 5 o n volte)
      private boolean singleRabinMiller(long p, long a, long b, long m) {
            long j = 0;
            long z = (((long)Math.pow(a,m)) % p);

            if (z==1 || z==p-1) return true;    // questo test e' superato

            while (true) {
                  if (j>0 && z==1) return false;      // di sicuro non e'
primo
                  j++;
                  if (j<b && z!=p-1) {
                        z = ((z*z) % p);
                  } else {
                        if (z==p-1) return true;      // questo test e'
superato
                        if (j==b && z!=p-1) return false;
                        break;
                  }
            }
            return false;
      }

      // Restituisce un numero primo casuale sufficientemente grande
      // (da 2^8 a 2^12, così il prodotto di 2 sara' da 2^16 a 2^24)
      public long getLongPrime() {
            return getLongPrime(260,4094);
      }
```

```
// Restituisce un numero primo casuale compreso tra min e max
public long getLongPrime(long min, long max) {
	long number;

	while (true) {
		number = min + (long)(Math.random()*(double)(max-min));

		if (number%2==0) number--;
		if (RabinMiller(number)) return number;
	}
}

// Inizializzazione dell'algoritmo RSA
// (torna { n, e, d } )
public long[] RSAstart() {
	long p,q,n,e,d;
	long temp[];
	long Murphy[] = { 0, 47, 64, 99, 126 };
	boolean OK = false;

	do {
		// Trova p, q, n
		p = getLongPrime();
		q = getLongPrime();
		n = p*q;

		// Calcola d, e
		while (true) {
			d = getLongPrime();
			temp = ExtendedEuclid(d,(p-1)*(q-1));
			if (temp[0]==1 && temp[1]!=0) {
```

```
                              e = temp[1];
                              break;
                        }
                  }

                  // Fa alcune prove anti-jella (Legge di Murphy)
                  OK = true;
                  for (int i=0; i<Murphy.length; i++) {
                        if
(Murphy[i]!=RSAdecryptLong(RSAencryptLong(Murphy[i],n,e),n,d))
                              OK = false;
                  }
            } while (!OK);

            // Fornisce in uscita { n, e, d }
            long out[] = { n, e, d };
            return out;
      }

      // Procedura per eseguire potenze intere in modulo
      // (svolge long = ((long)Math.pow(a,b) % n); senza overflow
      public long PowMod(long a, long b, long n) {
            if (a==1 || b==0) return 1;         // Casi banali
            if (a==0) return 0;
            if (n==0 || n==1) return 0;         // In caso di errore

            a = a % n;
            long result = a;

            // Eseguo (b-1) prodotti
            for (int i=1; i<b; i++) {
                  result = result * a;
```

```
            result = result % n;
        }
        return result;
    }

    // Algoritmo di cifratura RSA ( long -> long )
    private long RSAencryptLong(long plain, long n, long e) {

        return PowMod(plain,e,n);
    }

    // Algoritmo di decifratura RSA ( long -> long )
    private long RSAdecryptLong(long cipher, long n, long d) {

        return PowMod(cipher,d,n);
    }

    // Cifratura RSA byte[2] -> byte[3]
    private byte[] RSAencrypt2bytes(byte plain[], long n, long e) {
        if (plain.length!=2) return null;

        long cipher = RSAencryptLong(myDM.twoBytesToLong(plain),n,e);

        return myDM.longToThreeBytes(cipher);
    }

    // Decifratura RSA byte[3] -> byte[2]
    private byte[] RSAdecrypt3bytes(byte cipher[], long n, long d) {
        if (cipher.length!=3) return null;

        long plain = RSAdecryptLong(myDM.threeBytesToLong(cipher),n,d);
```

```
            return myDM.longToTwoBytes(plain);
      }

      // Cifratura RSA byte[] -> byte[] ( dim. *2 -> *3 )
      public byte[] RSAencrypt (byte plain[], long n, long e) {
            // Calcola il numero di blocchi da cifrare
            int numblocks = ((plain.length-1)/2+1);

            // Copia plain in nuovo array newplain di dimensione pari (n*2)
            byte newplain[] = new byte[numblocks*2];
            System.arraycopy(plain,0,newplain,0,plain.length);
            if (newplain.length>plain.length) newplain[newplain.length-
1]=0;

            // Crea array cipher di dimensione 3/2 di newplain
            byte cipher[] = new byte[numblocks*3];

            // Riempie tre alla volta gli elementi di cipher
            byte temp[] = new byte[2];
            for (int i=0; i<numblocks; i++) {
                  System.arraycopy(newplain,2*i,temp,0,2);

      System.arraycopy(RSAencrypt2bytes(temp,n,e),0,cipher,3*i,3);
            }
            return cipher;
      }

      // Decifratura RSA byte[] -> byte[] ( dim. *3 -> *2 )
      public byte[] RSAdecrypt (byte cipher[], long n, long d) {
            // Calcola il numero di blocchi da cifrare
            int numblocks = ((cipher.length-1)/3+1);
```

```
            // Copia cipher in nuovo array newcipher (dim=k*3)
            byte newcipher[] = new byte[numblocks*3];
            System.arraycopy(cipher,0,newcipher,0,cipher.length);
            if (newcipher.length>cipher.length) {
                  newcipher[newcipher.length-1]=0;
                  if (newcipher.length>cipher.length+1)
                        newcipher[newcipher.length-2]=0;
            }

            // Crea array plain di dimensione 2/3 di newcipher
            byte plain[] = new byte[numblocks*2];

            // Riempie due alla volta gli elementi di cipher
            byte temp[] = new byte[3];
            for (int i=0; i<numblocks; i++) {
                  System.arraycopy(newcipher,3*i,temp,0,3);

      System.arraycopy(RSAdecrypt3bytes(temp,n,d),0,plain,2*i,2);
            }
            return plain;
      }
}
```

21.4 La classe DataManager

```
/*
 * @(#)DataManager.java
 *
 * Classe di supporto per impacchettare e ricomporre i dati
 *
 * @author  Fabrizio Fazzino
 * @version 1.0         1996/XI/16
```

```
 */

import java.io.*;

public class DataManager {

      /* Quando voglio spedire delle stringhe nel socket,
      esse non vengono trasformate direttamente in bytes
      ma viene creata una struttura del seguente tipo:

            bytes[0] = n = numero di stringhe;
            bytes[1]...bytes[n] = lunghezza di ognuna delle n stringhe;
            bytes[n+1]...bytes[n+bytes[1]] = stringa 0;
            bytes[n+bytes[1]+1]...bytes[n+bytes[1]+bytes[2]] = stringa 1;
            ...
            bytes[n+bytes[1]+...+bytes[n-
1]+1]...bytes[n+bytes[1]+..+bytes[n]] =
                  = stringa n-1

      In tal modo la destinazione può facilmente compiere
      il procedimento inverso per ricavare le stringhe.

      Nel caso in cui i dati debbano essere cifrati, essi alla sorgente
      vanno prima impacchettati in array di byte, poi cifrati e spediti,
      e alla destinazione vanno decifrati e ricomposti come stringhe.

      N.B. Si vede che deve valere la proprietà:
            bytes.length == bytes[0]+bytes[1]+...+bytes[n]+1
      In pratica però mi accontento che sia >= , perché la cifratura
      potrebbe richiedere l'aggiunta di caratteri di riempimento.
      */
```

```
// Impacchetta le stringhe in array di byte con formattazione
// ( String[] -> byte[] )
public byte[] packStringsIntoBytes(String strings[]) {
      byte bytes[] = null;

      if (strings!=null && strings.length!=0) {
            int args = strings.length;
            int totalLength = 0;
            for (int s=0; s<args; s++)
                  totalLength += strings[s].length();

            bytes = new byte[1+args+totalLength];
            bytes[0] = (byte)args;
            for (int a=0; a<args; a++) {
                  int offset = 1+args;
                  for (int k=0; k<a; k++)
                        offset += strings[k].length();

                  bytes[a+1] = (byte)strings[a].length();

strings[a].getBytes(0,strings[a].length(),bytes,offset);
            }
      }

      return bytes;
}

// Ricompone l'array di stringhe originario
// ( byte[] -> String[] )
public String[] unpackStringsFromBytes(byte bytes[]) {
      String strings[] = null;
```

```
            if (bytes==null || bytes.length==0) {
                  return strings;
            } else {
                  int n = bytes[0];
                  if (bytes.length < n+1) {
                        return strings;
                  } else {
                        int utilBytes=1;
                        for (int z=0; z<=n; z++)
                              utilBytes += bytes[z];
                        if (utilBytes > bytes.length) {
                              return strings;
                        } else {
                              strings = new String[n];
                              for (int i=0; i<n; i++) {
                                    int count = (int)bytes[i+1];
                                    int offset=1;
                                    for (int x=0; x<=i; x++)
                                          offset += bytes[x];
                                    strings[i] = new
String(bytes,0,offset,count);
                              }
                        }
                  }
            }
            return strings;
      }

      // Metodi utili per la stampa

      // Stampa un array di boolean come sequenza di 0 e 1
      public void printBooleans(boolean array[]) {
```

```
            for(int z=0; z<array.length; z++)
                  System.out.print((array[z] ? 1 : 0));
            System.out.println();
      }

      // Stampa array di byte come numeri separati da virgole
      public void printBytes(byte array[]) {
            if(array!=null && array.length>0) {
                  System.out.print(array[0]);
                  for(int z=1; z<array.length; z++) {
                        System.out.print(","+array[z]);
                  }
                  System.out.println();
            }
      }

   // Indica se il numero e' dispari o meno:
   public boolean dispari(long numero) {
            return ((numero%2)==1);
      }

      // Metodi di conversione

      // Converte da long (es.hex) a boolean[64] (piu' significativo a sx)
   // Modifica: out[64], hex
   public void hexToBooleans(long hex, boolean out[]) {
       if ((hex>=0)&&(hex<=0x7fffffffffffffffL)) {
            for (int n=0; n<64; n++) {
                out[n] = dispari (hex >> (63-n));
            }
       } else {
            hex += (0x7fffffffffffffffL+1);
```

```
            out[0] = true;
            for (int n=1; n<64; n++) {
                out[n] = dispari (hex >> (63-n));
            }
        }
    }

    // Converte stringa di 0 e 1 in boolean[]
    public boolean[] stringToBooleans(String string) {
            boolean booleans[] = new boolean[string.length()];

            for (int i=0; i<string.length(); i++) {
                  if (string.charAt(i)=='0') booleans[i]=false;
                  else if (string.charAt(i)=='1') booleans[i]=true;
                  else return null;
            }
            return booleans;
    }

    // Converte boolean[] in stringa di 0 e 1
    public String booleansToString(boolean booleans[]) {
            StringBuffer string = new StringBuffer();

            for (int i=0; i<booleans.length; i++) {
                  if (booleans[i]) string.append('1');
                  else string.append('0');
            }
            return string.toString();
    }

    // Conversione singola byte -> boolean[8] usata dalla seguente
    public boolean[] byteToBooleans(byte input) {
```

```
        boolean output[] = new boolean[8];

        if (input<0) {
              output[0] = true;
              input = (byte)(128+(int)input);
        } else {
              output[0] = false;
        }
        for (int i=0; i<7; i++) {
              if (dispari(input >> i)) output[7-i]=true;
              else output[7-i]=false;
        }
        return output;
  }

  // Converte byte[] -> boolean[] (dim*8)
  public boolean[] bytesToBooleans(byte bytes[]) {
        boolean booleans[];

        if (bytes==null || bytes.length==0) return null;
        else booleans = new boolean[bytes.length*8];

        for (int s=0; s<bytes.length; s++) {

  System.arraycopy(byteToBooleans(bytes[s]),0,booleans,s*8,8);
        }

        return booleans;
  }

  // Conversione singola boolean[8] -> byte usata dalla seguente
  public byte booleansToByte(boolean input[]) {
```

```
            byte output = 0;

            for (int i=0; i<8; i++) {
                  if (input[i]) output += (byte)Math.pow(2,7-i);
            }
            return output;
      }

      // Converte boolean[] -> byte[] (dim/8 per eccesso)
      public byte[] booleansToBytes(boolean booleans[]) {
            byte bytes[];

            if (booleans==null || booleans.length==0) return null;
            else if(booleans.length%8==0) bytes = new
byte[booleans.length/8];
            else bytes = new byte[booleans.length/8+1];

            boolean temp[] = new boolean[8];
            for (int z=0; z<bytes.length; z++) {
                  System.arraycopy(booleans,8*z,temp,0,8);
                  bytes[z] = booleansToByte(temp);
            }

            return bytes;
      }

      // Converte long -> byte[2]
      public byte[] longToTwoBytes(long in) {
            byte out[] = new byte[2];

            out[1] = (byte)(in % 256);
            out[0] = (byte)((in/256) % 256);
```

```
            return out;
      }

      // Converte long -> byte[3]
      public byte[] longToThreeBytes(long in) {
            byte out[] = new byte[3];

            out[2] = (byte)(in % 256);
            out[1] = (byte)((in/256) % 256);
            out[0] = (byte)(((in/256)/256) % 256);

            return out;
      }

      // Converte long -> byte[8]
      public byte[] longToEightBytes(long in) {
            byte out[] = new byte[8];

            boolean sign = (in<0);              // conservo segno e lo
tolgo
            if (sign) in+=Long.MAX_VALUE+1;

            for (int i=0; i<8; i++) {
                  out[7-i] = (byte)( (in/(long)(Math.pow(256,i))) % 256 );
            }

            if (sign) out[0]+=(byte)128;  // rimetto il segno

            return out;
      }
```

```
	// Converte byte[2] -> long
	public long twoBytesToLong(byte in[]) {
		if (in.length!=2) return 0;

		int temp[] = new int[2];
		temp[0] = ( (in[0]>=0) ? (int)in[0] : ((int)in[0])+256 );
		temp[1] = ( (in[1]>=0) ? (int)in[1] : ((int)in[1])+256 );

		return (long)(temp[0]*256+temp[1]);
	}

	// Converte byte[3] -> long
	public long threeBytesToLong(byte in[]) {
		if (in.length!=3) return 0;

		int temp[] = new int[3];
		temp[0] = ( (in[0]>=0) ? (int)in[0] : ((int)in[0])+256 );
		temp[1] = ( (in[1]>=0) ? (int)in[1] : ((int)in[1])+256 );
		temp[2] = ( (in[2]>=0) ? (int)in[2] : ((int)in[2])+256 );

		return (long)(temp[0]*256*256+temp[1]*256+temp[2]);
	}

	// Converte byte[8] -> long
	public long eightBytesToLong(byte in[]) {
		if (in.length!=8) return 0;
		boolean sign = (in[0]<0);
		long out;

		if (sign) {
			in[0] += (byte)128;
			out = (-1)*(Long.MAX_VALUE+1);
```

```
            } else {
                  out = 0;
            }
            int temp[] = new int[8];
            for (int i=0; i<8; i++) {
                  temp[i] = ( (in[i]>=0) ? (int)in[i] : ((int)in[i])+256 );
                  out += temp[i]*(long)Math.pow(256,7-i);
            }

            return out;
      }

      // Converte String -> byte[]
      public byte[] stringToBytes(String in) {
            if (in.length()==0) return null;

            byte out[] = new byte[in.length()];

            // String.getBytes(int Str_start, int Str_end+1, byte[],
byte_start)
            in.getBytes(0,in.length(),out,0);

            return out;
      }

      // Converte byte[] -> String
      public String bytesToString(byte in[]) {
            if (in.length==0) return null;

            // Costr. String(byte[], int hi, int byte_start, int byte_tot)
            // oppure String(byte[], int hi)
            String out = new String(in,0);
```

```
            return out;
      }
}
```

22 - Il simulatore di esempio

22.1 Il simulatore Fab Net 3.0

Nella implementazione del Laboratorio Virtuale, come esempio, è stato incluso un simulatore di reti di computer. Tale simulatore, Fab Net 3.0, è un simulatore a riga di comando che permette di valutare, al variare di diversi parametri della rete, le prestazioni ed in particolare il ThroughPut e l'Access Delay ottenuti in funzione del WorkLoad presente sulla rete.

Il simulatore permette di variare il protocollo di comunicazione utilizzato nella rete simulata, scegliendone uno tra i tre protocolli a token, ovvero il Token Bus, il Token Ring e l'FDDI solitamente impiegato su reti in fibra ottica.

Nel seguito si riporta il sorgente C del simulatore.

22.2 Il file FABNET.C

```
/*************************************************************************
**/
/*                          >>>  Fab Net 3.0  <<<
*/
/*            Command line version - Written in Portable ANSI C
*/
/*                       (C) 1996 by Fabrizio Fazzino
*/
```

```
/*
*/
/*  Simulazione, valutazione e confronto delle prestazioni dei protocolli
*/
/*        TokenBus / TokenRing / FDDI  per le reti di calcolatori.
*/
/*************************************************************************
**/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#define TOKEN_BUS  0
#define TOKEN_RING 1
#define FDDI       2

/*************** PARAMETRI DELLA RETE INSERIBILI DALL'UTENTE
****************/
unsigned Protocol=TOKEN_BUS;  /* protocollo da adoperare per la simulazione
*/
unsigned Priority=0;      /* priorit… di cui valutare le prestazioni [0..3]
*/
unsigned Nodi=30;                                   /* numero di nodi */
float Percorso=1;                            /* distanza tra i nodi [Km] */
float Data_Rate=1;                      /* velocit… della sottorete [Mbps]
*/
unsigned long Lunghezza=1000;              /* lunghezza media messaggi [bit]
*/
float T_Simul=3000;           /* tempo virtuale di durata della simulazione
*/
```

```
float T_Pri[4]={50,40,30,15};        /* tempi di rotazione del token [msec]
*/

/********* VARIABILI AUSILIARIE PER IL CALCOLO DELLE PRESTAZIONI
***********/
float Time;                              /* variabile tempo globale [msec]
*/
float ThroughPut,TP_max;                          /* throughput [Mbps] */
float Access_Delay,AD_max ;                        /* ritardo [msec] */
float WorkLoad[4],WL_max;          /* WorkLoad di saturazione [frames/sec]
*/

struct nodo
{                               /* struttura di un singolo nodo della rete
*/
  float istante[4];               /* istante di generazione del messaggio
*/
  unsigned long lunghezza[4];                  /* lunghezza del messaggio
*/
  unsigned long msg_spediti[9];          /* num.msg.spediti dalla stazione
*/
  float last_token;                                  /* per calcolare TRT
*/
} *Coda;                            /* puntatore alla struttura della rete
*/

struct
{                 /* matrice per memorizzare i risultati delle simulazioni
*/
  float workload[9];          /* valori di WL a cui eseguire le simulazioni
*/
```

```
  float throughput[9][3];                          /* array di TP risultanti
*/
  float access_delay[9][3];                        /* array di AD risultanti
*/
  unsigned long intervallo_ritardi[9][3][16];/* contatore per distribuzione
*/
} Matrice;


/***************************** PROGRAMMA
*********************************/


/* SIMULA LA GENERAZIONE ESPONENZIALE DEGLI INTERARRIVI CON PARAMETRO "WL".
   FORNENDO INFATTI IL WORKLOAD "WL" [frames/sec] RESTITUISCE LA VARIABILE
   TEMPORALE "EXP" [msec] CON DISTRIBUZIONE ESPONENZIALE. */
float Exp(float WL)
{
  float Var;
  Var=1000*log((double)((1/(1-((float)((long)rand())/(RAND_MAX+1))))))/WL;
  return Var;
}


/* DEVE ESSERE ESEGUITA PER INIZIALIZZARE LE OPPORTUNE VARIABILI PRIMA DI
   UNA QUALUNQUE SIMULAZIONE IN CONDIZIONI PREFISSATE; AD ESEMPIO ESEGUENDO
   LA CLASSICA CAMPAGNA DI SIMULAZIONI TALE PROCEDURA DOVRA' ESSERE
ESEGUITA
   PER OGNUNO DEGLI 8 PUNTI. */
void Point_Reset(void)
{
  unsigned Indice,Livello;


  for(Indice=0;Indice<Nodi;Indice++,Coda++)         /* inizializza le code
*/
```

```
  {
    for(Livello=0;Livello<4;Livello++)
    {
      (*Coda).istante[Livello]=Exp(WorkLoad[Livello]);

(*Coda).lunghezza[Livello]=floor((double)((Lunghezza/2+Lunghezza*((long)ran
d())/(RAND_MAX+1))));
    }
    (*Coda).last_token=0;
  }
  Coda-=Nodi;
}

/* INIZIALIZZA LE VARIABILI CHE RICHIEDONO DI ESSERE RESETTATE SOLO PRIMA
   DI OGNI CAMPAGNA DI SIMULAZIONI. */
void Screen_Reset(void)
{
  unsigned Punto,Banda,Colonna,Indice;

  for(Punto=0;Punto<9;Punto++)
  {
    Matrice.workload[Punto]=0;

    for(Banda=0;Banda<3;Banda++)
    {
      Matrice.throughput[Punto][Banda]=0;
      Matrice.access_delay[Punto][Banda]=0;

      for(Colonna=0;Colonna<16;Colonna++)
      Matrice.intervallo_ritardi[Punto][Banda][Colonna]=0;
    }
```

```
    for(Indice=0;Indice<Nodi;Indice++,Coda++)
      (*Coda).msg_spediti[Punto]=0;
    Coda-=Nodi;
  }
}

/* SIMULA IL RELATIVO PROTOCOLLO TOKEN BUS, E OLTRE ALLE VARIBILI GLOBALI
   (COI DATI DELLA RETE) HA BISOGNO CHE LE SIANO PASSATE LE INFORMAZIONI
   RELATIVE ALLA ATTUALE SIMULAZIONE DA COMPIERE, IN MODO DA POTERE
   VISUALIZZARE CORRETTAMENTE I RISULTATI DELLE SIMULAZIONI.
   "Priority" INDICA INFATTI IL LIVELLO DI PRIORITA' DI CUI VALUTARE LE
   PRESTAZIONI, "Punto" INDICA DI QUALE PUNTO DA INTERPOLARE SI STIA
   EFFETTUANDO LA SIMULAZIONE, "Curva" INDICA L'OCCUPAZIONE DEI LIVELLI
   A PRIORITA' SUPERIORE (SIMULO UNA OCCUPAZIONE DEL 25%, 50%, 75% DELLA
   BANDA MA SOLO NEL CASO IN CUI "Priority" NON SIA LA MASSIMA). */
void TokenBus(unsigned Punto, unsigned Curva)
{

                        /* con carico [frames/sec]. */
  float Start;
  float sum_ritardi=0;
  float TRT,THT;
  unsigned Indice,Livello;
  unsigned long num_ritardi=0;
  unsigned colonna;
  unsigned long bit_trasmessi=0;

  THT=T_Pri[0]/(float)Nodi;
  Time=T_Pri[0];          /* per ogni simulazione resetto il tempo globale
*/
  while(Time<T_Pri[0]+T_Simul)                    /* tempo di simulazione
*/
```

```
  {
    for(Indice=0;Indice<Nodi;Indice++,Coda++)          /* scandisce i nodi
*/
    {
      Time+=12*8/(1000*Data_Rate)+Percorso/400;             /* trasm.token
*/
      TRT=Time-(*Coda).last_token;              /* calcola TRT ultimo giro
*/
      (*Coda).last_token=Time;                /* aziona timer prossimo giro
*/

      Start=Time;                           /* fa partire il timer del nodo
*/
      for(Livello=0;Livello<4;Livello++)        /* scandisce le 4 priorit…
*/
      {
      while( ((*Coda).istante[Livello]<Time) &&
             ((Livello==0 && Time-Start<THT)
            ||(Livello!=0 && TRT<T_Pri[Livello])) )
      {                         /* se ha token + coda piena => trasmette
*/
        if(Livello==Priority)                       /* aggiorna parametri
*/
        {
          sum_ritardi+=Time-(*Coda).istante[Priority];         /* per AD
*/
          num_ritardi++;

          bit_trasmessi+=(*Coda).lunghezza[Priority]-12*8;     /* per TP
*/

          if(Time-(*Coda).istante[Priority]<1) colonna=0;
```

```
          else colonna=floor((double)((4*log10((double)((Time-
(*Coda).istante[Priority])))))));
          if(colonna>=16) colonna=15;
          Matrice.intervallo_ritardi[Punto][Curva][colonna]++; /*ritardi*/
          (*Coda).msg_spediti[Punto]++;                        /*Fairness*/
        }
               /* aumenta tempo globale e TRT del tempo di trasmissione */
        Time+=((float)(*Coda).lunghezza[Livello])/(1000*Data_Rate)+
             Percorso/200;
        TRT +=((float)(*Coda).lunghezza[Livello])/(1000*Data_Rate)+
             Percorso/200;

                                    /* riempie daccapo la coda */

(*Coda).lunghezza[Livello]=floor((double)((Lunghezza/2+Lunghezza*((long)ran
d())/(RAND_MAX+1))));
        (*Coda).istante[Livello]+=Exp(WorkLoad[Livello]);
      }
      }
    }
    Coda-=Nodi;
  }

  if (num_ritardi!=0) Access_Delay=sum_ritardi/(float)num_ritardi;
    else Access_Delay=0;                                  /* AD=[frames/sec]
*/
  ThroughPut=((float)bit_trasmessi)/(1000*Time);
/*TP=[Mbps]*/
}

/* SIMULA IL PROTOCOLLO TOKEN RING UTILIZZANDO LE STESSE VARIABILI DEL
   PRECEDENTE CASO TOKEN BUS, MA E' ALQUANTO PIU' LABORIOSA E LENTA,
```

```
   IN QUANTO AD OGNI PASSAGGIO DI TOKEN BISOGNA DETERMINARE QUALE E' IL
   MESSAGGIO A PRIORITA' PIU' ELEVATA TRA QUELLI PRESENTI IN TUTTE LE
   STAZIONI. */
void TokenRing(unsigned Punto, unsigned Curva)
{
  float Start;
  float sum_ritardi=0;
  float THT;
  unsigned Indice,IndPrior,Livello;
  unsigned long num_ritardi=0;
  unsigned colonna;
  unsigned long bit_trasmessi=0;
  unsigned Prior=3,Rent;
  unsigned Nodo_Impilatore[3],Prior_Precedente[3];

  THT=T_Pri[0]/(float)Nodi;
  Time=T_Pri[0];           /* per ogni simulazione resetto il tempo globale
*/
  while(Time<T_Pri[0]+T_Simul)                    /* tempo di simulazione
*/
  {
    for(Indice=0;Indice<Nodi;Indice++,Coda++)
    {
      Time+=3*8/(1000*Data_Rate)+Percorso/((float)Nodi*200); /* trasm.token
*/
      Start=Time;                          /* fa partire il timer del nodo
*/

           /* calcola la priorit… pi— elevata dei messaggi in coda (Rent)
*/
      Rent=3;
      Coda-=Indice;
```

```
      for(IndPrior=0;IndPrior<Nodi;IndPrior++,Coda++)
      {
      for(Livello=0;Livello<Rent;Livello++)
      {
        if((*Coda).istante[Livello]<Time)
        {
          Rent=Livello;
          break;
        }
      }
      if(Prior==0) break;
      }
      Coda-=IndPrior;
      Coda+=Indice;
                  /* confronta la priorit… attuale con quella prenotata */
      if(Rent<Prior)          /* aumenta la priorit… e impila la precedente
*/
      {
      Nodo_Impilatore[Rent]=Indice;
      Prior_Precedente[Rent]=Prior;
      Prior=Rent;
      }
      if((Rent>Prior)&&(Indice==Nodo_Impilatore[Prior]))
      {   /* la stazione che aveva impilato riabbassa alla prior.precedente
*/
      Prior=Prior_Precedente[Prior];
      }

      for(Livello=0;Livello<Prior+1;Livello++)
      {
      while(((*Coda).istante[Livello]<Time)&&(Time-Start<THT))
      {                       /* se ha token + coda piena => trasmette */
```

```
        if(Livello==Priority)
        {                                          /* aggiorna parametri
*/
          sum_ritardi+=Time-(*Coda).istante[Priority];          /* per AD
*/
          num_ritardi++;

          bit_trasmessi+=(*Coda).lunghezza[Priority]-13*8;      /* per TP
*/

          if(Time-(*Coda).istante[Priority]<1) colonna=0;
          else colonna=floor((double)((4*log10((double)((Time-
(*Coda).istante[Priority])))))));
          if(colonna>=16) colonna=15;
          Matrice.intervallo_ritardi[Punto][Curva][colonna]++; /*ritardi*/
          (*Coda).msg_spediti[Punto]++;                        /*Fairness*/
        }
                  /* aumenta il tempo globale del tempo di trasmissione */
        Time+=((float)(*Coda).lunghezza[Livello])/(1000*Data_Rate);
                                    /* riempie daccapo la coda */

(*Coda).lunghezza[Livello]=floor((double)((Lunghezza/2+Lunghezza*((long)ran
d())/(RAND_MAX+1))));
        (*Coda).istante[Livello]+=Exp(WorkLoad[Livello]);
      }
      }
      Time+=Percorso/200;         /* tempo richiesto per arrivo tutte frame
*/
    }
    Coda-=Nodi;
  }
```

```
  if (num_ritardi!=0) Access_Delay=sum_ritardi/(float)num_ritardi;
    else Access_Delay=0;                                  /* AD=[frames/sec]
*/
  ThroughPut=((float)bit_trasmessi)/(1000*Time);
/*TP=[Mbps]*/
}

/* SIMULA IL PROTOCOLLO FDDI IN MANIERA MOLTO SIMILE AL TOKEN BUS. */
void FDDI_Protocol(unsigned Punto, unsigned Curva)
{
                                          /* con carico [frames/sec]. */
  float sum_ritardi=0;
  float TRT;
  unsigned Indice,Livello;
  unsigned long num_ritardi=0;
  unsigned colonna;
  unsigned long bit_trasmessi=0;

  Time=T_Pri[0];           /* per ogni simulazione resetto il tempo globale
*/
  while(Time<T_Pri[0]+T_Simul)                      /* tempo di simulazione
*/
  {
    for(Indice=0;Indice<Nodi;Indice++,Coda++)
    {
      Time+=11*8/(1000*Data_Rate)+Percorso/((float)Nodi*200); /*
trasm.token*/
      TRT=Time-(*Coda).last_token;              /* calcola TRT ultimo giro
*/
      (*Coda).last_token=Time;               /* aziona timer prossimo giro
*/
```

```
      for(Livello=0;Livello<4;Livello++)     /* scandisce 4 code a priorit…
*/
      {                                  /* meccanismo a soglia ( T_Pri[] )
*/
      while(((*Coda).istante[Livello]<Time)&(TRT<T_Pri[Livello]))
      {                       /* se ha token + coda piena => trasmette */
        if(Livello==Priority)
        {                                          /* aggiorna parametri
*/
          sum_ritardi+=Time-(*Coda).istante[Priority];          /* per AD
*/
          num_ritardi++;

          bit_trasmessi+=(*Coda).lunghezza[Priority]-21*8;      /* per TP
*/

          if(Time-(*Coda).istante[Priority]<1) colonna=0;
          else colonna=floor((double)((4*log10((double)((Time-
(*Coda).istante[Priority])))))));
          if(colonna>=16) colonna=15;
          Matrice.intervallo_ritardi[Punto][Curva][colonna]++; /*ritardi*/
          (*Coda).msg_spediti[Punto]++;                        /*Fairness*/
        }
               /* aumenta tempo globale e TRT del tempo di trasmissione */
        Time+=((float)(*Coda).lunghezza[Livello])/(1000*Data_Rate);
        TRT+=((float)(*Coda).lunghezza[Livello])/(1000*Data_Rate);

                                    /* riempie daccapo la coda */

(*Coda).lunghezza[Livello]=floor((double)((Lunghezza/2+Lunghezza*((long)ran
d())/(RAND_MAX+1))));
        (*Coda).istante[Livello]+=Exp(WorkLoad[Livello]);
```

```
      }
      }
    }
    Coda-=Nodi;
  }

  if (num_ritardi!=0) Access_Delay=sum_ritardi/(float)num_ritardi;
    else Access_Delay=0;                                  /* AD=[frames/sec]
*/
  ThroughPut=((float)bit_trasmessi)/(1000*Time);
/*TP=[Mbps]*/
}

/* E' LA FUNZIONE CHE IN BASE ALLE SCELTE INIZIALI DA MENU PROVVEDE A
   SELEZIONARE IL PRESCELTO PROTOCOLLO DA SIMULARE.
   I PARAMETRI DA PASSARE SONO GLI STESSI DELLE TRE PROCEDURE TOKEN BUS,
   TOKEN RING E FDDI, COSI' COME DESCRITTI ALL'INIZIO DEL TOKEN BUS;
   E' ALTRESI' PRESENTE UN ULTERIORE PARAMETRO "PROTOCOL" CHE PROVVEDE
   APPUNTO A SELEZIONARE IL PROTOCOLLO PRESCELTO DALL'UTENTE. */
void Protocollo(unsigned Punto, unsigned Curva)
{
  Point_Reset();                    /* setta istante,lunghezza e last_token
*/

  switch(Protocol)
  {
    case TOKEN_BUS: TokenBus(Punto,Curva); break;
    case TOKEN_RING: TokenRing(Punto,Curva); break;
    case FDDI: FDDI_Protocol(Punto,Curva); break;
  }
}
```

```
/* QUESTA FUNZIONE E' IL VERO CERVELLO DELLA SIMULAZIONE, IN QUANTO
   EFFETTUA LA CAMPAGNA DI SIMULAZIONI CHIAMANDO LA FUNZIONE
   Protocollo E MEMORIZZANDO TUTTI I RISULTATI NELLA STRUTTURA Matrice.*/
void Simulatore(void)
{
  unsigned Punto,Curva,Setta_WL;
  time_t t;

  Coda=(struct nodo *) calloc(Nodi,sizeof(struct nodo));
  if(Coda==NULL)
  {
    printf("# Insufficient memory (try with less nodes).\n");
    exit(EXIT_FAILURE);
  }

  srand((unsigned) time(&t));
  Screen_Reset();

  if(Priority==0) WL_max=2*1e6*Data_Rate/(float)(Nodi*Lunghezza);
  else WL_max=1e6*Data_Rate/(float)(Nodi*Lunghezza);
  for(Setta_WL=0;Setta_WL<4;Setta_WL++) WorkLoad[Setta_WL]=1;

  printf("# Fab Net 3.0 is simulating ");

  for(Punto=1;Punto<9;Punto++)
  {
    Matrice.workload[Punto]=WL_max*((float)Punto)/8;
    WorkLoad[Priority]=Matrice.workload[Punto];
    if(Priority==0)
    {
      Protocollo(Punto,0);
      Matrice.throughput[Punto][0]=ThroughPut;
```

```
      Matrice.access_delay[Punto][0]=Access_Delay;


      printf("...");
    }
    else for(Curva=0;Curva<3;Curva++)           /* 3 curve banda occupata
*/
    {
      WorkLoad[0]=WL_max*0.25*(float)(Curva+1); /* carico del 25%,50%,75%
*/
      Protocollo(Punto,Curva);
      Matrice.throughput[Punto][Curva]=ThroughPut;
      Matrice.access_delay[Punto][Curva]=Access_Delay;


      printf(".");
    }
  }
  printf("\n");


  free(Coda);
}


/* VISUALIZZA LA CORRETTA SINTASSI. */
void PrintSyntax(void)
{
    printf("\n              >>>  Fab Net 3.0  <<<\n");
    printf("Command line version - Written in portable ANSI C\n");
    printf("(C) 1996 by Fabrizio Fazzino, ffazzino@k200.cdc.unict.it\n");
    printf("Token Bus / Token Ring / FDDI  Computer Networks
Simulator\n\n");
    printf("Syntax is the following : \n");
    printf("  fabnet PROTOCOL PRIORITY NODES PATH DATA_RATE LENGTH ");
    printf("T_SIMUL [TRT(4)..TRT(1)]\n\n");
```

```
    printf("Where:  ( the default '*' is used if parameter doesn't match
)\n");
    printf("      PROTOCOL = int 0=TokenBus, 1=TokenRing, 2=FDDI ( * = 0
)\n");
    printf("      PRIORITY = int [number] of priority 4...1 ( * = 4 )\n");
    printf("      NODES = int [number] of nodes ( * = 30 )\n");
    printf("      PATH = float [Km] between two nodes ( * = 1 )\n");
    printf("      DATA_RATE = float [Mbps] data-rate ( * = 1 )\n");
    printf("      LENGTH = long [bit] mean messages length ( * = 1000
)\n");
    printf("      T_SIMUL = float [msec] virtual time to simulate ( * =
3000 )\n");
    printf("      T_PRI = 4 float [msec] tokenrotationtimes ( * = 50 40 30
15 )\n");
}

/* VISUALIZZA I RISULTATI SULLO STANDARD OUTPUT. */
void PrintResults(void)
{
    int i,j;
    int curve;

    /* Primo record con gli 11 parametri utilizzati in ingresso */
    printf("#%u,%u,%u,%g,%g,",Protocol,4-Priority,Nodi,Percorso,Data_Rate);
    printf("%u,",Lunghezza);
    printf("%g,",T_Simul);
    printf("%g,%g,%g,%g\n",T_Pri[0],T_Pri[1],T_Pri[2],T_Pri[3]);

    /* Secondo record con valori max su cui normalizzare i grafici */
    TP_max=Data_Rate;
    if(Priority==0) AD_max=Matrice.access_delay[4][0];
    else AD_max=Matrice.access_delay[4][1];
```

```
    if(AD_max<100) AD_max=100;
    printf("#%g,%g\n",TP_max,AD_max);

    /* Terzo record con ascisse WorkLoad ( float workload[9] ) */
    printf("#");
    for(i=1;i<=7;i++) printf("%g,",Matrice.workload[i]);
    printf("%g\n",Matrice.workload[8]);

    /* Uno o tre record per il ThroughPut ( float throughput[9][3] ) */
    if(Priority==0) curve=1;
    else curve=3;
    for(j=0;j<curve;j++)
    {
      printf("#");
      for(i=1;i<=7;i++) printf("%g,",Matrice.throughput[i][j]);
      printf("%g\n",Matrice.throughput[8][j]);
    }

    /* Uno o tre record per l'Access Delay ( float access_delay[9][3] ) */
    for(j=0;j<curve;j++)
    {
      printf("#");
      for(i=1;i<=7;i++) printf("%g,",Matrice.access_delay[i][j]);
      printf("%g\n",Matrice.access_delay[8][j]);
    }

/*
  float workload[9];          /* valori di WL a cui eseguire le simulazioni
  float throughput[9][3];                         /* array di TP risultanti
  float access_delay[9][3];                       /* array di AD risultanti
  unsigned long intervallo_ritardi[9][3][16];/* contatore per distribuzione
*/
```

```
}

/* PROCEDURA PRINCIPALE CHE VISUALIZZA LA SINTASSI CORRETTA OPPURE
   AVVIA LA SIMULAZIONE COI PARAMETRI RICEVUTI E STAMPA I RISULTATI. */
int main(int argc, char *argv[])
{
  unsigned protocol,priority,nodi;
  float percorso,data_rate;
  unsigned long lunghezza;
  float t_simul;
  float t_pri[4];

  if(argc!=8 && argc!=12) /* Num. di parametri sbagliato */
  {
    PrintSyntax();
    exit(EXIT_FAILURE);
  }
  else  /* Setto tutti i parametri */
  {
    protocol = (unsigned)atol(argv[1]);
    if(protocol<=2) Protocol=protocol;

    priority = (unsigned)atol(argv[2]);
    if(priority>=1 && priority<=4) Priority=4-priority;

    nodi = (unsigned)atol(argv[3]);
    if(nodi>=1) Nodi=nodi;

    percorso = (float)atof(argv[4]);
    if(percorso>0) Percorso=percorso;

    data_rate = (float)atof(argv[5]);
```

```
  if(data_rate>0) Data_Rate=data_rate;

  lunghezza = (unsigned long)atol(argv[6]);
  if(lunghezza>0) Lunghezza=lunghezza;

  t_simul = (float)atof(argv[7]);
  if(t_simul>0) T_Simul=t_simul;

  if(argc==12)
  {
    t_pri[0] = (float)atof(argv[8]);
    if(t_pri[0]>0) T_Pri[0]=t_pri[0];

    t_pri[1] = (float)atof(argv[9]);
    if(t_pri[1]>0) T_Pri[1]=t_pri[1];
    if(T_Pri[1]>T_Pri[0]) T_Pri[1]=T_Pri[0];

    t_pri[2] = (float)atof(argv[10]);
    if(t_pri[2]>0) T_Pri[2]=t_pri[2];
    if(T_Pri[2]>T_Pri[1]) T_Pri[2]=T_Pri[1];

    t_pri[3] = (float)atof(argv[11]);
    if(t_pri[3]>0) T_Pri[3]=t_pri[3];
    if(T_Pri[3]>T_Pri[2]) T_Pri[3]=T_Pri[2];
  }

  /* Avvio la simulazione */
  Simulatore();

  /* Visualizzo i risultati */
  PrintResults();
}
```

```
  exit(EXIT_SUCCESS);
  return 0;
}
```

Bibliografia

[WHIT] James Gosling e Henry McGilton, **'The Java Language Environment: A White Paper'**, Technical Report, Sun Microsystems, 1995 (disponibile presso `http://java.sun.com/whitePaper.Gosling_McGilton.html`)

[LANG] Sun Microsystems, **'The Java Language Specification'**, Technical Report, 1995 (disponibile presso `http://sunsite.kth.se/javaspec_1.html`)

[JTUT] Mary Campione e Kathy Walrath, **'The Java Tutorial'**, Technical Report, Sun Microsystems, 1996 (disponibile presso `http://java.sun.com/index.html`)

[PROG] Sun Microsystems, **Basic & Advanced Java Programming**, 1995

[JSEC] Joseph A. Bank, **'Java Security'**, Technical Report, Massachusetts Institute of Technology, 1995 (disponibile presso `http://swissnet.ai.mit.edu/javapaper.html`)

[FIRE] W.Cheswick e S.Bellovin, **Firewalls and Internet Security**, Addison-Wesley, 1995

[DBSE] S.Castano, M.Fugini, G.Martella e P.Samarati, **Database Security**, Addison-Wesley, 1994

[SCHN] Bruce Schneier, **Applied Cryptography** (2nd edition), John Wiley & Sons, 1995

[TANE] Andrew S. Tanenbaum, **Modern Operating Systems**, Prentice-Hall, 1992

[STEV] Richard W. Stevens, **Unix Network Programming**, Prentice-Hall, 1990

[STAL] William Stallings, **Network and Internetwork Security**, Prentice-Hall, 1995

Sommario