UNIVERSITÀ DEGLI STUDI DI PARMA
Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Studio e Sperimentazione di un Client VNC basato su HTML5

Study and Experimentation of a VNC Client based on HTML5

Relatore
Chiar.mo Prof. Federico Bergenti

Candidato
Andrea Casarotto

Anno Accademico 2012/2013

Ai miei genitori,
Emanuela e Alessandro

Indice

Prefazione

Lo studio di questa tesi porrà al centro della propria attenzione un Client *VNC*, un particolare tipo di software che permette la connessione remota tra due computer che hanno accesso ad Internet. Nello specifico, tale Client è integrato all'interno di una pagina web sviluppata con la tecnologia *HTML5*.

L'esistenza delle connessioni remote ricopre un'importanza primaria all'interno di una realtà fortemente informatizzata e digitalizzata. Questo tipo di collegamento permette di accedere a macchine situate geograficamente a molti chilometri di distanza da quella su cui effettivamente si opera, spesso in una diversa nazione o addirittura in un diverso continente.

I motivi per cui si riscontra la necessità di dover stabilire una connessione remota sono molteplici, infiniti. Sia in ambito lavorativo, operando su macchinari o programmi che richiedono un intervento tecnico da parte di un informatico specializzato, sia in ambito accademico, con lo studio delle tecnologie passate e la ricerca di sempre nuove sperimentazioni orientate al futuro, che in ambito privato, nei casi in cui ci si deve collegare al computer di casa mentre si è lontani.

Il lavoro che segue sarà strutturato in modo tale da evidenziare l'importanza dei Client *VNC* fornendo al lettore tutti gli strumenti e gli elementi necessari per inoltrarsi al meglio ed in modo esaustivo in questo tipo di studio.

Il primo capitolo della tesi fornirà una panoramica generale sulle *Macchine Virtuali*: che cosa sono, in cosa consistono, quali importanti vantaggi apportano nel campo dell'Informatica e quali sono le modalità di accesso che permettono l'interazione tra *Macchina Virtuale* e computer ospitante. Verranno poi forniti pochi ed esaustivi esempi delle principali *Macchine Virtuali* che hanno rappresentato un fattore determinante nello sviluppo dell'Informatica degli ultimi anni.

Il secondo capitolo tratterà al dettaglio uno dei protocolli usati nella comunicazione remota tra due computer: il Protocollo *RFB*. Verrà spiegato

l'utilizzo che se ne potrà fare in un contesto di comunicazione remota e grafica tra due dispositivi e verranno illustrate le modalità di comunicazione tra le parti che determinano i capi saldi del collegamento, Client e Server. Nello specifico verranno presentati tutte le fasi che compongono il Protocollo ed i tipi di messaggi che i due computer in comunicazione si scambieranno per poter stabilire la connessione.

Il terzo ed ultimo capitolo conterrà il corpo centrale della vera e propria sperimentazione di cui si fa oggetto questa tesi. Saranno presentati ed ampiamente spiegati gli strumenti utilizzati per lo studio preso in esame: la Macchina Virtuale *Bochs*, gli strumenti di sviluppo software *GWT* ed il Client *VNC*. Verranno inoltre esibiti alcuni semplici e specifici esempi che mostreranno, per mezzo di porzioni di codice di programmazione e di illustrazioni grafiche, la struttura e l'effettiva esecuzione del progetto.

Farà seguito un capitolo finale che trarrà alcune conclusioni in merito ai risultati ottenuti dallo studio e dalla sperimentazione del Client *VNC* preso in esame ed offrirà una serie di vantaggi riguardanti il suo possibile impiego.

Capitolo 1

Le Macchine Virtuali

Al giorno d'oggi esiste una vastissima gamma di computer, spesso anche completamente diversi tra loro: workstation aziendali, notebook portatili, tablet manageriali e smartphone sono solo alcuni pochi esempi di ciò che il mercato del ventunesimo secolo offre riguardo all'informatica ed all'elettronica. Tuttavia, per quanto diversi possano essere tra loro tali dispositivi, sia a livello di componenti fisiche con cui sono stati creati ed assembrati, sia a livello di applicazioni software, posseggono tutti una struttura base che li accomuna.

Tale struttura, comune a tutti i dispositivi elettronici che rientrano, in un certo senso, nella classificazione di computer, riguarda la loro progettazione fortemente stratificata che suddivide idealmente le diverse componenti di un sistema in un certo numero di livelli o strati a seconda della loro utilità e del loro fine: questa progettazione prende il nome di **Struttura a Livelli**.

Ogni livello è un oggetto astratto che incapsula dati e le operazioni che trattato tali dati, ed è composto da strutture dati e procedure richiamabili dai livelli superiori facenti parte del sistema.

Di conseguenza, il livello più basso corrisponde all'hardware, seguito poi dal livello kernel e da tutti gli altri livelli che formano il sistema per terminare, infine, nell'ultimo livello che si occupa di definire e gestire l'interfaccia utente.

La necessità di operare spesso con molti tipi differenti di sistemi, rispetto a quello presente su una macchina, ha portato alla definizione del concetto di **Macchina Virtuale** (*Virtual Machine*), un particolare software che opera un'astrazione ed una virtualizzazione di tutte le componenti di un computer e che crea un ambiente virtuale che emula il comportamento di una macchina fisica, comprensiva di tutte le sue componenti hardware quali processore, memorie e dispositivi, garantendo una normale assegnazione delle risorse ed esecuzione di software, quali ad esempio i sistemi operativi, come se ognuno di essi girasse concretamente su un vero e proprio computer fisico.

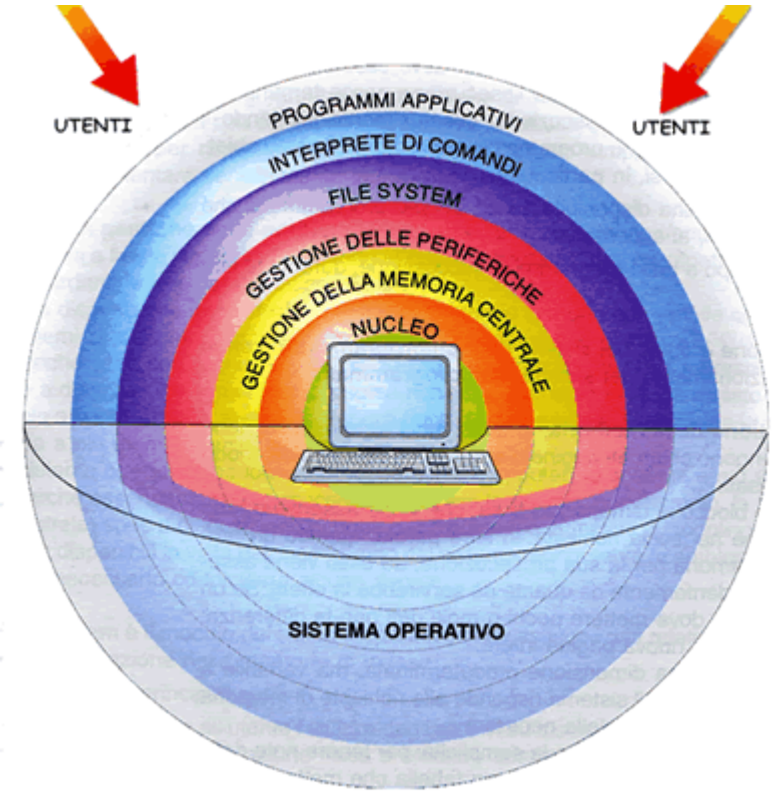

Figura 1.1: Struttura a Livelli

1.1 Vantaggi

La possibilità di disporre di più ambienti di esecuzione sullo stesso computer, astraendo dall'interfaccia hardware sottostante, fornisce ottime ragione per adottare questa strategia di lavoro, nonchè numerosi vantaggi legati all'utilizzo delle risorse e del sistema.

Un primo vantaggio nell'utilizzo di una *Macchina Virtuale* consiste nell'alto ed efficace livello di sicurezza fornito dalla macchina stessa: la possibilità di operare in un ambiente che virtualizza completamente l'hardware, l'accesso alle risorse ed i processi di un sistema, nonchè il completo isolamento tra le diverse macchine presenti su uno stesso computer, si pone a salvaguardia della struttura fisica da lui astratta e virtualizzata aumentando così il suo grado di sicurezza a fronte di eventuali rischi che si possono presentare durante la normale esecuzione di un'applicazione o l'accesso a dati che potrebbero compromettere il software.

La creazione di diversi ambienti virtuali per mezzo delle *Macchine Virtuali*, permette inoltre ad ogni processo di essere dotato di un proprio processore e di un proprio spazio di memoria virtuale riservato, suddividendo idealmente il computer originario in una pluralità di macchine differenti su cui possono operare diversi sistemi operativi ed applicazioni. La capacità delle *Macchine Virtuali* di garantire una perfetta astrazione dell'hardware concede, di conseguenza, la facoltà di operare e svolgere tutte le attività necessarie alla normale e corretta esecuzione di un processo o sistema operativo, come le chiamate di sistema ed un vero e proprio file system virtuale su cui eseguire ogni tipo di operazione.

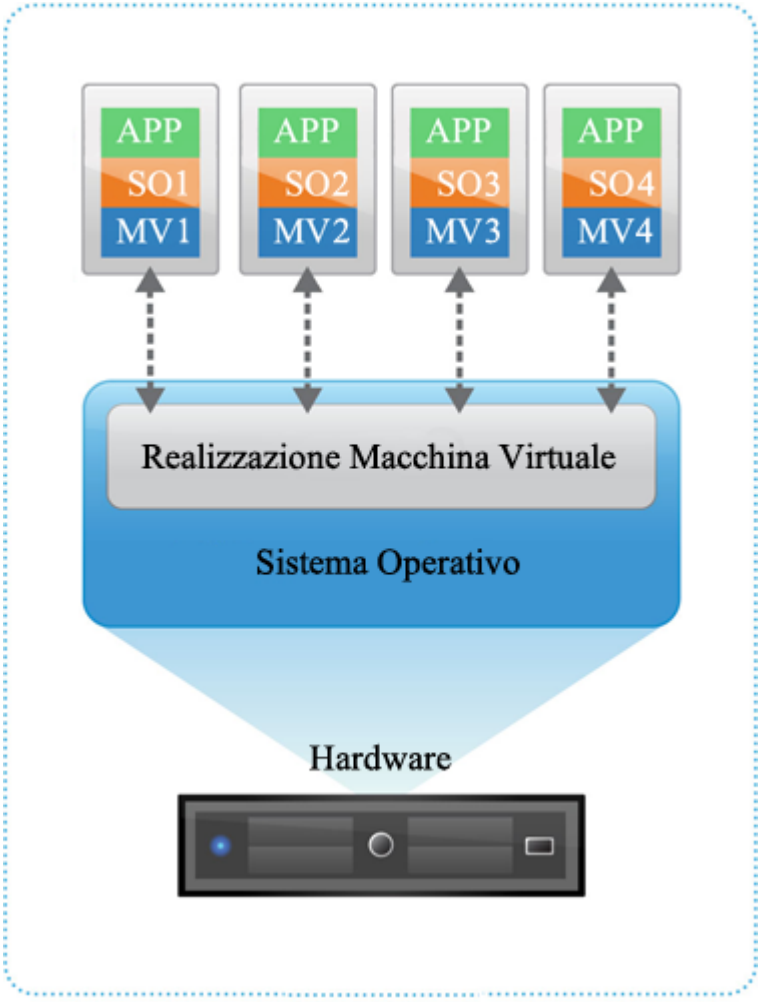

Figura 1.2: Struttura di una Macchina Virtuale

Altra rilevante caratteristica che dona una particolare importanza all'utilizzo delle *Macchine Virtuali*, consiste nel fatto che questi tipi di software rappresentano un mezzo perfetto per la ricerca e lo sviluppo dei sistemi operativi: a causa della loro complessità, i sistemi operativi sono programmi talmente vasti e complessi, che una piccola modifica in una sua parte specifica potrebbe comportare il malfunzionamento o la generazione di complicati errori di programmazione in un'altra diversa parte del programma. Per questo motivo, oltre al fatto che un sistema operativo lavora ed opera in modalità di sistema, quindi non direttamente sotto il controllo dell'utente, interessando e coinvolgendo nella sua esecuzione tutta la macchina, ogni modifica apportata ed errata potrebbe seriamente compromettere il suo funzionamento. Si rivela quindi necessario prestare la massima attenzione ogni volta che si decide di operare modifiche che manomettono, anche in un modo apparentemente lieve ed insignificante, il sistema operativo. Ulteriore difficoltà nello studio e nello sviluppo dei sistemi operativi consiste nell'obbligo di fermare l'esecuzione del sistema in fase di modifica e controllo, condizione necessaria implicata dall'importanza che assume un sistema operativo come programma che si frappone tra l'hardware di un computer ed il software messo a disposizione per l'utenza.

Per questi motivi, le *Macchine Virtuali* rappresentano un'importante innovazione nello studio dettagliato dei sistemi operativi: dal momento che ogni *Macchina Virtuale* rappresenta, idealmente, un singolo computer, è possibile eseguire su di essa un sistema operativo secondario, rispetto a quello attivo sulla macchina fisica, sul quale apportare ogni tipo di modifica che in nessun modo comprometterà l'integrità e l'utilizzo del sistema operativo installato sul computer su cui opera e viene eseguita la *Macchina Virtuale*.

1.2 Esempi

Ultimamente, anche grazie all'importante vantaggio di poter eseguire su un'unica macchina fisica più sistemi operativi anche incompatibili tra loro o con le componenti hardware cui è formato il computer, le *Macchine Virtuali* hanno riscontrato un sempre più ampio utilizzo e sviluppo. Di seguito sono riportati esempi delle principali *Macchine Virtuali* che hanno conosciuto una recente diffusione.

1.2.1 VMware

VMware è una nota applicazione sviluppata da una importante società americana, la *VMware Inc.* ([4]), che si occupa dello studio e dello sviluppo di software per la realizzazione di macchine virtuali per i sistemi Intel x86. VMware funge da applicazione in un qualsiasi sistema operativo ospitante, permettendo quindi l'installazione e l'esecuzione di alcuni sistemi operativi ospiti, eseguiti in modo concorrente sulla stessa macchina, ognuno dei quali agisce come macchina virtuale separata ed indipendente, dotata di una propria CPU virtuale ed altri elementi virtuali come memorie, dischi, interfacce di rete, dispositivi e via di seguito.

1.2.2 Java Virtual Machine

La Macchina Virtuale Java (*Java Virtual Machine, JVM*) è uno speciale software prodotto dalla *Sun Microsystems* ([6]), implementato ed ospitato su un sistema operativo residente, che esegue programmi scritti in Java. Questa *Macchina Virtuale* rappresenta per lo più un calcolatore astratto che consiste di un caricatore delle classi, di cui sono formati i programmi Java, e da un interprete del linguaggio che esegue il *bytecode*, codice del linguaggio di macchina della JVM indipendente dall'architettura sottostante e fornito dal compilatore che ha compilato il codice delle classi del programma. Il caricatore carica le classi che compongono il programma e le mette a disposizione dell'interprete, il quale le esegue dopo che i controlli sulla correttezza del codice delle classi hanno avuto esito positivo. La JVM gestisce inoltre la memoria a sua disposizione in modo automatico ed efficiente, procedendo alla sua ripulitura per mezzo del *garbage collection*.

1.2.3 Bochs

Bochs è un software open source, scritto in C++ e sviluppato da *Kevin Lawton* ([8]), che emula architetture x86 e AMD64. Questa speciale *Macchina Virtuale* può essere ospitata su qualsiasi sistema operativo e fornisce una virtualizzazione ed una emulazione di processori Intel x86, dispositivi I/O, memorie, dischi, display, collegamenti Ethernet e BIOS oltre alle più comuni periferiche hardware disponibili: tutto ciò lo rende in grado di eseguire i principali sistemi operativi in circolazione, Linux, DOS, Windows, etc., caratteristica che conferisce a questa *Macchina Virtuale* un'ampia varietà di utilizzo permettendo all'utente di eseguire sistemi operativi e software all'interno dell'emulatore installato sul proprio computer, come se avesse a disposizione una pluralità di macchine diverse, con relativo software, raccol-

te in un unico computer. Per maggiori dettagli su questo tipo di *Macchina Virtuale*, si rimanda alla consultazione del capitolo **??**.

1.3 Accesso alla Macchina Virtuale

L'introduzione delle *Macchine Virtuali* è solo un piccolo tassello di ciò che rappresenta l'enorme sviluppo che è avvenuto ed avviene tutt'ora, nel mondo dell'informatica e della comunicazione digitale, la cui espansione capillare ha facilmente raggiunto, interessato ed inglobato ogni ambito lavorativo e sociale, professionale e privato. Casi lampanti ed evidenti di questa grande informatizzazione si presentano, a chiunque, giornalmente: basti pensare, ad esempio, a studi commercialisti ed aziende che basano la propria contabilità su programmi gestionali che coinvolgono ogni parte della struttura di una società, dalla più ordinaria gestione di un magazzino alla difficile e complessa redazione e compilazione di bilanci e registrazioni economiche; i complessi sistemi informatici delle strutture sanitarie ed ospedaliere, spesso talmente vasti ed estesi da ricoprire e coinvolgere più centri, laboratori ed ambulatori, edifici che non difficilmente sono situati anche a chilometri di distanza l'uno dall'altro in città diverse; gli importanti programmi e software di gestione finanziaria degli istituti di credito e bancari, nei quali la sicurezza dei dati e la loro integrità e correttezza assumono un ruolo estremamente importante, oltre al fatto da dover sempre aver accesso alla rete mondiale di internet sì da garantire una situazione aggiornata in tempo reale sui cambiamenti dei titoli di borsa e dei valori monetari che interessano le valute internazionali, la cui frequenza raggiunge l'ordine del minuto, se non addirittura del secondo.

A fronte di questa realtà, si evince chiaramente come tutti questi sistemi informatici siano strettamente collegati tra loro, per mezzo di Internet, in una connessione mondiale che crea una rete talmente vasta e di proporzioni così ampie e globali da raggiungere ogni parte del mondo senza il minimo sforzo o difficoltà. Si afferma quindi il bisogno estremo di una connessione virtuale che permetta di usufruire di tutti i vantaggi rappresentati da ciò che comporta l'esistenza di una grande rete mondiale di questo livello, e che al tempo stesso garantisca un certo margine di sicurezza a fronte di innumerevoli rischi e pericoli che si possono presentare, imprevisti che non sono rappresentati soltanto da attacchi ed intrusioni al sistema da parte di hacker o cracker, ma che possono nascere e presentarsi dalla normale e quotidiana esecuzione del software specializzato nel lavoro di cui si occupa un particolare istituto o società.

In un tale scenario, le connessioni virtuali tra dispositivi e sistemi diversi tra loro, spesso situati anche a notevoli distanze geografiche, assumono un ruolo importante che coadiuva il lavoro stesso dei programmi che ne richiedono l'intervento. Sono stati quindi studiati ed introdotti particolari software che permettono la creazione di questi collegamenti virtuali per mezzo di una rete di trasmissione e di un protocollo di comunicazione. Due esempi di questi software sono rappresentati da *VNC* (*Virtual Network Computing*) e dai *Client con Protocollo RDP*.

1.3.1 VNC

Virtual Network Computing, comunemente noto con l'acronimo *VNC*, è un software che permette la creazione e la gestione di una connessione virtuale e remota tra due computer. Affinché il collegamento si possa stabilire, deve essere necessaria la presenza di una connessione tra i due dispositivi sulla cui rete ci si appoggia per la trasmissione dei dati e dei pacchetti. Il software *VNC* consta di tre componenti particolari:

- *Client VNC*: Il Client VNC è rappresentato dal computer da cui un utente può stabilire la connessione ad un altro computer remoto. Il software *VNC* del Client si occupa di inviare la richiesta ad un altro computer e di gestire graficamente il flusso di dati ricevuti in modo tale da offrire all'utente del Client un'interfaccia grafica che gli permetta di operare virtualmente come se fosse situato fisicamente al computer verso cui avviene la connessione. Per poter stabilire la connessione è obbligatorio che l'utente fornisca al software l'indirizzo (*host*) e la porta (*port*) del computer remoto in modo tale da dotare il programma di tutte le informazioni per localizzare il computer remoto sulla rete ed effettuare il collegamento con esso. Nei casi in cui è richiesta autenticazione da parte dell'utente che desidera stabilire la connessione, allora dovrà essere fornita anche una password;

- *Server VNC*: Il Server VNC è rappresentato dal computer remoto a cui un utente si connette. Questo apparecchio può essere situato in una posizione geografica diversa e lontana da quella in cui è localizzato il Client, oppure può essere virtualizzato da una *Macchina Virtuale* installata sul Client stesso. Unico requisito è che abbia libero accesso ad una rete di trasmissione dei dati, come ad esempio Internet. Su questo computer deve essere operativo ed eseguito in background il software *VNC* relativo al lato server che si pone in attesa ed in ascolto di eventuali richieste di connessione da parte dei Client. Non appena il

Server riceve una richiesta di collegamento, insieme al Client provvederà a definire e stabilire tutti i parametri necessari, compatibili ad entrambe le parti, utilizzati per la comunicazione;

- *Protocollo RFB*: Il Protocollo RFB (*Remote Framebuffer*) consiste in un insieme di regole e specifiche che definiscono le diverse fasi con cui viene stabilito un collegamento tra Client e Server e le modalità di comunicazione durante la connessione remota. Per maggiori dettagli si rimanda alla consultazione del capitolo 2.

Dopo questa panoramica sulle connessioni virtuali, si evince come sia semplice ed immediato effettuare un collegamento tra due dispositivi elettronici: è sufficiente aver a disposizione un computer, il software *VNC* ed una connessione ad internet per poter idealmente stabilire un collegamento remoto tra il proprio computer Client ed un computer Server. Tuttavia è necessario esprimere alcune considerazioni specifiche che riguardano attentamente, e da vicino, la natura di questo collegamento.

La connessione remota tra due computer deve garantire e permettere un lavoro efficace ed efficiente, dal punto di vista delle risorse impiegate per stabilire e gestire il collegamento, dei tempi di risposta che intercorrono da una richiesta alla sua effettiva esecuzione, dei consumi di memoria locale e della banda sul mezzo di trasmissione utilizzato, in modo tale che l'intervento effettuato durante il collegamento tra Client e Server si svolga con maggior rapidità e miglior precisione possibili.

Per tali motivi, una connessione remota e virtuale tra due computer deve rispettare determinate caratteristiche:

- *Protezione*: la connessione deve garantire il più alto livello possibile di sicurezza, in modo tale da evitare assolutamente il rischio di possibili intrusioni esterne che potrebbero inficiare e compromettere la validità dell'operato. E' necessario inoltre che si dia l'opportunità al Server di richiedere da colui che cerca di stabilire la connessione di dimostrare e verificare la sua identità attraverso un sicuro sistema di autenticazione e controllo, sì da evitare l'ingresso ad utenti che non sono autorizzati. Questo tipo di caratteristica assume un ruolo di rilevante importanza in tutti quei casi in cui la connessione remota avviene verso Server che gestiscono e trattano dati sensibili e privati, come gli istituti bancari, studi di commercialisti od aziende ospedaliere;

- *Stabilità*: la connessione deve stabilirsi, tra Client e Server, in modo stabile e solido, per permetter loro di operare in modo fluido e diretto senza subire continuamente interruzioni del collegamento con tutti i rischi che ciò comporta, come lo smarrimento e la perdita di dati ed aggiornamenti che non sono ancora giunti a destinazione e quindi applicati. Una connessione stabile è necessaria in tutti i casi in cui si devono gestire ingenti flussi di dati, come i filmati, brani audio od immagini ad alta risoluzione, i quali devono essere in grado di giungere a destinazione senza dispersioni o rallentamenti sul mezzo di trasporto, durante la connessione, per poter essere considerati integri e corretti;

- *Velocità*: la comunicazione sulla rete di trasporto, durante la connessione remota tra Client e Server, deve poter avvenire in modo veloce e rapido, senza subire ingenti ritardi o gravi interruzioni. Una buona velocità di trasmissione può essere garantita dal corretto uso del mezzo di trasporto e della banda che si ha a disposizione per la comunicazione; l'utilizzo corretto ed oculato di queste risorse garantisce un'operatività in tempo reale, requisito necessario in tutti quei casi in cui è richiesta tempestività d'intervento dove la rapidità di ricevimento, esecuzione e processamento dei dati ricopre un ruolo di primaria importanza, in mancanza della quale vi è il rischio di compromettere il risultato e la qualità del lavoro svolto;

- *Portabilità*: è necessario che la comunicazione avvenga nei confronti di ogni tipo di computer e dispositivo elettronico, senza che il marchio di produzione, l'hardware, il sistema operativo od il software applicativo rappresentino un ostacolo o, peggio, una barriera invalicabile che impedisce il collegamento. L'utilizzo di questo software e la comunicazione deve essere il più ampio e generalizzato possibile, senza restrizioni e senza particolari requisiti che potrebbero rendere inaccessibile un computer. L'universalità dei metodi utilizzati per la definizione delle regole di connessione e la loro applicazione garantisce la massima libertà di comunicazione senza dover sottostare a vincoli di marchi, brevetti o case di produzione.

1.3.2 Client Protocollo RDP

Il Protocollo *RDP* (*Remote Desktop Protocol*) è un protocollo di comunicazione sviluppato da Microsoft che permette la comunicazione remota tra due computer, in modalità grafica, tramite il *Microsoft Terminal Services* (*MTS*).

Il Protocollo *RDP* è un protocollo multicanale, che consente quindi di disporre di diversi e numerosi canali virtuali di comunicazione, per mezzo dei quali è possibile effettuare il trasporto dei dati, lo scambio di informazione, la comunicazione con le periferiche, la trasmissione di dati crittografati e di eventi di I/O. Inoltre, sebbene offra oltre 64.000 canali distinti per la trasmissione dei dati, attualmente per le normali comunicazioni ne utilizza soltanto un singolo canale, che provvede a trasmettere e gestire ogni tipo di dato ed evento.

Il Protocollo *RDP* è stato sviluppato in modo da essere completamente indipendente dallo strato di trasporto su cui si appoggia: per questo motivo, oltre a supportare molteplici tipologie di rete e protocolli, potrà essere in futuro estesa con l'aggiunta di ulteriori driver di trasporto e protocolli di comunicazione senza compromettere il funzionamento del sistema o dover intervenire con modifiche ed interventi per adattarne la compatibilità alle nuove tecnologie e sviluppi. Al momento, la versione attuale del Protocollo *RDP* si basa su TCP/IP.

Le attività svolte con il Protocollo *RDP* riguardano prevalentemente la comunicazione tra due o più computer che formano la connessione remota: tra di essi è possibile scambiare messaggi ed informazioni sottoforma di pacchetti incapsulati e fatti transitare sulla rete sottostante in modo tale da essere indirizzati al proprio destinatario dopo essere stati crittografati e compattati. La comunicazione è bidirezionale, ed i dati in risposta effettuano lo stesso percorso dei dati precedenti, in modo inverso.

Nonostante sia stato sviluppato da Microsoft, il Protocollo *RDP* permette di effettuare comunicazioni e stabilire connessioni anche da sistemi operativi diversi da Windows, come nel caso di Mac OS X grazie ad uno speciale Client per Mac che si occupa di gestire al meglio questo tipo di comunicazione tra sistemi differenti.

Capitolo 2

Il Protocollo RFB

Quando due o più computer devono entrare in comunicazione tra di loro è necessario che si stabilisca un collegamento, fisico o virtuale che sia, tra di essi. Le regole e le convenzioni usate in questa comunicazione sono globalmente note e definite con il termine di protocolli.

Un **protocollo** è quindi un accordo, stabilito tra le due parti, sul modo in cui deve procedere la comunicazione.

Il protocollo utilizzato nella connessione remota tra due computer, ed in particolar modo in un *VNC* (*Virtual Network Computing*), prende il nome di Protocollo **RFB** (*Remote Framebuffer*).

2.1 Introduzione

RFB è un semplice protocollo che definisce e gestisce le regole di accesso remoto ad un'interfaccia grafica. Poiché questo protocollo richiede un numero strettamente limitato di requisiti per colui che ne usufruisce, può venire facilmente utilizzato su una vasta gamma di macchine con differenti strutture hardware, oltre che ad essere ampiamente applicabile con qualsiasi sistema operativo od applicazione.

Il computer remoto dal quale un utente effettua il collegamento viene solitamente chiamato **RFB Client**: su questa postazione verranno visualizzati i dati e le immagini ricevute dalla connessione stabilita, nonché si avrà la possibilità di apportare modifiche ed interventi per mezzo di una tastiera e di un puntatore mouse. Il Client avrà effettuato un collegamento con un computer solitamente situato in una locazione diversa e lontana. Questo computer prende il nome di **RFB Server**: sul sistema operativo e le applicazioni di

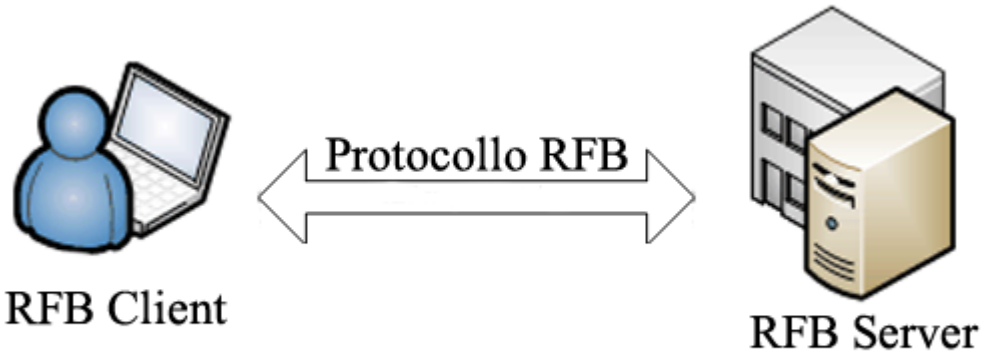

Figura 2.1: Sistema di comunicazione VNC

tale computer verranno eseguite le modifiche operate dal Client.

Il Protocollo *RFB*, dal momento che non prevede l'assegnazione di uno stato per il Client che effettua la connessione, permette una certa libertà di accesso svincolata da una ben determinata sessione: se un Client si disconnette da un Server, interrompendo la connessione che successivamente viene ripristinata, lo stato dell'interfaccia grafica dell'utente viene preservato come se non si fosse verificata nessuna interruzione.

Non vi è nessun vincolo od obbligo in merito alla postazione fisica da cui effettuare la connessione; il protocollo permette la massima libertà e mobilità: ovunque sia disponibile una connessione ad Internet, un utente può eseguire l'accesso, per mezzo del Protocollo *RFB*, alle proprie applicazioni.

2.1.1 Protocollo Display

La parte del Protocollo *RFB* relativa alla visualizzazione delle immagini ricevute, si basa su una semplice primitiva grafica: *"disegna un rettangolo di pixel in una data posizione X, Y"*. Per quanto questa specifica possa apparentemente sembrare un metodo inefficiente ed inefficace per disegnare svariati componenti di una stessa interfaccia grafica, il protocollo accetta e concede diversi metodi di codifica dei dati pixel che formano l'immagine, concedendo quindi un ampio grado di flessibilità di scambio e gestione dei parametri quali, ad esempio, la larghezza di banda della rete (*network bandwith*), la velocità di disegno da parte del Client e la velocità del Server di processare

le richieste ricevute.

Una sequenza di rettangoli forma un *Framebuffer Update* (o più semplicemente *Update*), che di fatto rappresenta un cambiamento dello stato di un *Framebuffer* con un altro, in modo simile alla sequenza dei frame di immagine che formano un video.

Il protocollo di aggiornamento e passaggio dei vari *Update* si basa sulla richiesta effettuata dal Client: un *Update* è inviato dal Server al Client soltanto dopo aver ricevuto una richiesta esplicita da parte di quest'ultimo. Questa caratteristica dona al protocollo *RFB* una forte ed importante adattività che gli permette di gestire al meglio ogni tipo di connessione e Client: se un Client o una rete risultano particolarmente lenti, ad esempio, basso sarà il tasso d'invio degli *Update*, in modo tale da permettere al ricevente di processare il tutto senza congestionare il collegamento e provocare ulteriori e gravi rallentamenti che potrebbero influire negativamente sulla connessione.

2.1.2 Protocollo di Input

La parte del Protocollo *RFB* relativa all'input dei dati e dei segnali si basa sul modello classico di workstation formato da una tastiera ed un dispositivo dotato di uno o più pulsanti, come ad esempio un mouse o altri dispositivi di I/O. Gli eventi di input sono inviati al Server dal Client ogni qualvolta viene premuto un tasto o viene tracciato lo spostamento del puntatore del mouse.

2.1.3 Rappresentazione dei pixel

La prima interazione tra il Client ed il Server include una negoziazione del *formato* e della *codifica* con i quali i dati pixel verranno inviati. Scopo di tale negoziazione è quello di rendere il lavoro per il Client il più facile ed immediato possibile, garantendo non solo una certa efficacia d'azione, assicurata dalla certezza, per il Server, di inviare al Client dati secondo un formato che possa essere processato ed eseguito, ma anche una notevole efficienza, scegliendo il formato e la codifica che permettono al Client di svolgere il miglior lavoro impiegando il minor tempo possibile.

Il *formato* dei pixel è la rappresentazione dei colori individuali per ogni valore pixel. I formati più comunemente utilizzati sono 24-bit o 16-bit *true colour*, dove i bit di ogni valore pixel impostano direttamente le intensità di rosso, verde e blu, e 8-bit *colour map* in cui un'arbitraria mappatura viene

usata per tradurre le intensità RGB direttamente dai valori pixel.

La *codifica* si riferisce a come un rettangolo di pixel deve venire inviato. Tale specifica viene inviata insieme ai dati relativi ai rettangoli da disegnare nella finestra del Client, che sono quindi preceduti dalle coordinate X, Y della posizione del rettangolo sullo schermo, la sua lunghezza ed altezza (*width, height*) ed il tipo di codifica utilizzata (*encoding type*).

Il Protocollo *RFB* definisce alcuni tipi di codifica da applicare, a seconda delle situazioni che si presentano, ma può essere esteso aggiungendo altri tipi di codifica dei dati. I tipi di codifica standard definiti dal Protocollo *RFB* sono Raw, Copy Rectangle, RRE, Hextile e ZRLE descritti nella *sezione 2.3.5*.

2.2 Estensioni al Protocollo

Essendo il Protocollo *RFB* facilmente adattabile ad ogni tipo di hardware, sistema ed applicazione, esiste una pluralità di modi con cui è possibile estenderlo per ottenere il massimo dell'efficienza a seconda dei mezzi a propria disposizione.

Nuovi tipi di codifiche possono essere aggiunti al protocollo mantenendo la compatibilità tra i Server ed i Client attualmente esistenti. Nell'eventualità che una codifica non sia supportata da un Server, essa viene automaticamente ignorata in favore di un'altra codifica standard definita dal protocollo. A loro volta, i Client non richiederanno mai codifiche che non sono in grado di gestire.

Pseudo codifiche possono affiancare le codifiche vere e proprie: tali tipi di codifica possono essere richieste da alcuni Client per dichiarare ad un Server che essi supportano un certo tipo di codifica che estende il protocollo. Se una certa estensione non è supportata dal Server, viene ignorata. Per maggiori dettagli si veda la *sezione 2.3.6*.

Nuovi tipi di sicurezza aggiunti al Protocollo *RFB* concedono un'ulteriore flessibilità al comportamento che il protocollo assume in determinati casi senza sacrificare la compatibilità tra i Client ed i Server esistenti. Client e Server che concordano gli stessi tipi di sicurezza possono comunicare tra loro senza dover necessariamente far uso del medesimo protocollo.

2.3 Messaggi del protocollo

Il Protocollo *RFB* opera su ogni tipo di rete di trasporto, sia quelle orientate ai bit, che quelle orientate ai messaggi, anche se convenzionalmente viene usato per mezzo di una connessione TCP/IP. Esistono tre fasi specifiche di protocollo:

1. La fase di *Handshaking* ha lo scopo di far concordare al Client ed al Server la versione del Protocollo *RFB* utilizzato ed il tipo di sicurezza da adottare.

2. La fase di *Inizializzazione* permette alle due parti di scambiarsi messaggi di inizializzazione per poter stabilire effettivamente la connessione remota.

3. La fase di *Scambio*, infine, è la fase in cui avvengono le normali interazioni di scambio dei messaggi dal Client al Server e viceversa.

La descrizione del Protocollo *RFB* che segue fa uso dei tipi base U8, U16, U32 (*Unsigned Integer 8, 16 e 32 bit*) e S8, S16, S32 (*Signed Integer 8, 16 e 32 bit*). Tutti i byte Integer sono in *big endian* (il byte più significativo è il primo ad essere ricevuto).

2.3.1 Messaggi di Handshaking

Versione del protocollo

La fase di Handshaking ha inizio con l'invio al Client, da parte del Server, di un messaggio contenente il numero di versione del Protocollo *RFB* supportato ed utilizzato di default dal Server. Il Client risponderà poi con un messaggio con cui comunicherà al Server l'effettiva versione del protocollo da utilizzare per stabilire la connessione.

Le versioni attualmente in uso sono le seguenti: 3.3, 3.7, 3.8.

Estensioni particolari applicate al Protocollo *RFB* non modificano la sua versione fino a quando possono essere ignorate nell'eventualità che non siano supportate da una delle due parti.

Il messaggio della versione del protocollo è formato da una stringa di caratteri ASCII che rispetta il formato *"RFB xxx.yyy\n"* dove *xxx* ed *yyy* rappresentano i numeri maggiori e minori di versione.

Nel caso della versione 3.8 del Protocollo *RFB*:

Num.Byte	Valore
12	"RFB 003.008\n" (hex 52 46 42 20 30 30 33 2e 30 30 38 0a)

Sicurezza

Una volta stabilita la versione del Protocollo *RFB* da utilizzare, Server e Client decidono su che tipo di sicurezza applicare nello scambio dei messaggi.

Dalla versione 3.7 in avanti, il Server elenca tutti i tipi di sicurezza (*security-types*) che è in grado di supportare:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*numbers-of-security-types*
numbers-of-security-types	U8 array	*security-types*

Se nell'elenco che il Server invia compare almeno un tipo di sicurezza supportato dal Client, esso ritorna al Server un singolo byte che concerne il numero del tipo di sicurezza da adottare per la connessione:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*security-types*

Se il numero del tipo di sicurezza (*number-of-security-types*) è zero, allora per qualche ragione (una delle quali potrebbe essere l'impossibilità di supporto del numero di versione del Protocollo *RFB*) la connessione è fallita ed il messaggio è subito seguito da un altro che contiene il motivo del fallimento della connessione (*reason-string*) la cui lunghezza (*reason-length*) è specificata e fissata:

Num.Byte	Tipo (Valore)	Descrizione
4	U32	*reason-length*
reason-length	U8 array	*reason-string*

Dopo aver inviato il motivo del fallimento della connessione (*reason-string*), il Server chiude la connessione.

Il tipo di sicurezza (*security-type*) qui definito è il seguente:

Numero	Nome
0	*Invalid*
1	*None*
2	*VNC Authentication*

Esito della sicurezza

Una volta che il *security-type* è stato concordato e questa fase di Handshaking giunge al termine, il protocollo continua la comunicazione inviando un messaggio contenente l'esito delle operazione di sicurezza effettuate (*SecurityResult*).

Num.Byte	Tipo (Valore)	Descrizione
4	U32	*status*
	0	*OK*
	1	*failed*

Se l'esito è positivo ed il controllo è stato eseguito con successo, allora il Protocollo *RFB* passa alla fase di Inizializzazione (sezione 2.3.2).

In caso di insuccesso, il Server invia al Client una stringa contenente il motivo per il quale è avvenuto il fallimento (*reason-string*) e successivamente chiude la connessione:

Num.Byte	Tipo (Valore)	Descrizione
4	U32	*reason-length*
reason-length	U8 array	*reason-string*

Tipi di sicurezza

I tipi di sicurezza definiti dal Protocollo *RFB*, da concordare in fase di Handshaking, sono i seguenti:

- **Nessuno.** Se nessun tipo di sicurezza è richiesto, in fase di stabilimento della connessione, ed i dati devono essere inviati senza alcun tipo di codifica, il Protocollo *RFB* continua inviando il messaggio contenente l'esito dell'operazione (sezione 2.3.1).

- **Autenticazione VNC.** Se come controllo di sicurezza è richiesta un'autenticazione *VNC*, con relativa codifica dei dati pixel, il Server invia un messaggio casuale di 16 byte:

Num.Byte	Tipo (Valore)	Descrizione
16	U8	*challenge*

Il Client decodifica il messaggio ricevuto dal Server con l'algoritmo crittografico *DES* (*Data Encryption Standard*) usando la password inserita dall'utente in fase di login ed invia una risposta al Server:

Num.Byte	Tipo (Valore)	Descrizione
16	U8	*response*

Successivamente il Protocollo *RFB* continua inviando il messaggio contenente l'esito dell'operazione (sezione 2.3.1).

2.3.2 Messaggi di Inizializzazione

Quando la fase di Handshaking termina con successo, ed il Client e Server possono comunicare in modo sicuro e certi che entrambe le parti siano compatibili nell'applicazione del Protocollo *RFB*, ha inizio la fase di Inizializzazione in cui viene stabilita la connessione.

Inizializzazione del Client

Il Client invia al Server un messaggio con cui tenta di stabilire l'esclusività della connessione remota: in base al valore del flag inviato nel messaggio, il Server sarà in grado di determinare se condividere il desktop con tutti i Client connessi in quel momento (*shared-flag* = true) oppure se deve concedere l'accesso esclusivo al Client con cui tenta di stabilire il collegamento forzando la disconnessione con tutti gli altri Client connessi (*shared-flag* = false).

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*shared-flag*

Inizializzazione del Server

Dopo aver ricevuto il messaggio di inizializzazione del Client, il Server risponde con un suo messaggio in cui indica al Client la dimensione del *Framebuffer*, il formato dei pixel ed il nome associato al desktop:

Num.Byte	Tipo (Valore)	Descrizione
2	U16	*framebuffer-width*
2	U16	*framebuffer-height*
16	PIXEL_FORMAT	*server-pixel-format*
4	U32	*name-length*
name-length	U8 array	*name-string*

dove PIXEL_FORMAT è:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*bits-per-pixel*
1	U8	*depth*
1	U8	*big-endian-flag*
1	U8	*true-colour-flag*
2	U16	*red-max*
2	U16	*green-max*
2	U16	*blue-max*
1	U8	*red-shift*
1	U8	*green-shift*
1	U8	*blue-shift*
3		*padding*

Server-pixel-format specifica il formato utilizzato dai pixel del Server, a meno di una specifica richiesta di adottare un formato diverso da parte del Client. Tale richiesta verrà eseguita dal Client per mezzo di uno specifico messaggio (sezione 2.3.3).

Bits-per-pixel corrisponde al numero di bit usati per ogni valore dei pixel trasmessi nel collegamento e non deve essere inferiore dell'intensità (*depth*) che indica il numero di bit utilizzabili nei valori dei pixel. I valori supportati per *bits-per-pixel* sono 8, 16 o 32.

Se *true-colour-flag* è impostato con un valore diverso da zero, i successivi sei elementi che compongono il messaggio specificano come estrarre le intensità dei colori rosso, verde e blu dal valore dei pixel. *Max* indica il valore

massimo di un colore (= 2^n - 1 con n numero di bit utilizzati per quel colore); *Shift* indica il numero di spostamenti necessari per ottenere il valore del colore in un pixel dal bit meno significativo. Se *true-colour-flag* è impostato con il valore zero, il Server utilizzerà valori di pixel che non sono formati dalle intensità di colori canonici rosso, verde e blu, ma fornirà un indice con cui consultare una fissata mappa dei colori (*SetColourMapEntries*) specificata in un messaggio successivo (sezione 2.3.4).

2.3.3 Messaggi del Client

Ultimata la fase di Inizializzazione e stabilita la connessione tra le due parti, Client e Server iniziano il vero e proprio scambio di messaggi che determina il corpo della comunicazione remota. Il Protocollo *RFB* prevede un certo numero di messaggi predefiniti, ma come già detto in precedenza è possibile applicare un'estensione alle regole di comunicazione, che per essere utilizzate devono ricevere effettiva conferma di compatibilità, sia dal Server che dal Client. Di seguito sono riportati i messaggi predefiniti del protocollo inviati dal Client al Server.

Formato dei pixel

Questo tipo di messaggio imposta il formato dei valori dei pixel inviati nei messaggi di *FramebufferUpdate*. Se un Client non richiede alcun tipo di formato pixel, il Server applicherà il formato specificato nel messaggio di Inizializzazione del Server (sezione 2.3.2).

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	0	*message-type*
3			*padding*
16	PIXEL_FORMAT		*pixel-format*

dove PIXEL_FORMAT è descritto nella sezione 2.3.2.

Codifica dei pixel

Imposta il tipo di codifica da applicare ai valori dei pixel da inviare al Server. I tipi di codifica specificati nel messaggio sono elencati con una priorità: il loro ordinamento (descrescente) indica al Server quali tipi di codifica dei pixel il Client predilige rispetto ad altri. Se nessun tipo di codifica è specificato, allora viene automaticamente applicata la codifica *RAW* (2.3.5).

Oltre ai canonici tipi di codifica specificati nel Protocollo *RFB*, un Client può richiedere l'utilizzo di una pseudo codifica per dichiarare al Server che lui fa utilizzo di altri tipi di codifica che estendono il protocollo. Se il Server non supporta le estensioni del protocollo richieste dal Client, tali specifiche vengono ignorate e viene applicata un tipo di codifica supportata da entrambe le parti.

Per ulteriori approfondimenti sui tipi di codifica definiti dal protocollo si veda la sezione 2.3.5. Per approfondimenti sulle pseudo codifiche si fa rimando alla sezione 2.3.6.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	2	*message-type*
1			*padding*
2	U16		*number-of-encodings*

Segue l'elenco dei numeri di codifica:

Num.Byte	Tipo	(Valore)	Descrizione
4	S32		*encoding-type*

Richieste di aggiornamento del Framebuffer

Questo messaggio informa il Server che il Client è interessato all'area del *Framebuffer* determinata nel pacchetto da specifiche coordinate (*x-position* e *y-position*) e dimensioni (*width* e *height*). Ad una richiesta di aggiornamento del *Framebuffer* (*FramebufferUpdateRequest*) il Server risponde con un singolo aggiornamento del *Framebuffer* (*FramebufferUpdate*), in quanto assume che il Client tenga copia di tutte le parti del *Framebuffer* di cui interessato, motivo per cui il Server invia al Client soltanto le parti del *Framebuffer* che hanno effettivamente subito un cambiamento specificato da un valore numerico incrementale (*incremental*). Se il Client necessita di ricevere nuovamente l'intero *Framebuffer*, sarà per lui sufficiente impostare tale valore a zero sì da informare il Server che è in attesa di ricevere l'intero contenuto dell'area del buffer.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	3	*message-type*
1	U8		*incremental*
2	U16		*x-position*
2	U16		*y-position*
2	U16		*width*
2	U16		*height*

Eventi da tastiera

Questi eventi si verificano al momento della pressione di un tasto. Quando il tasto è premuto, *down-flag* viene impostato con un valore diverso da zero; quando il tasto è rilasciato, il valore di *down-flag* ritorna a zero.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	4	*message-type*
1	U8		*down-flag*
2			*padding*
4	U32		*key*

Per il valore dei tasti si utilizza la codifica standard in valori ASCII. Per maggiori informazioni in merito si consiglia di consultare The Xlib Reference Manual pubblicato da O'Reilly & Associates ([5]).

Eventi da puntatore

Questi eventi tracciano lo spostamento del puntatore sullo schermo (*x-position* e *y-position*) nonché la pressione dei tasti del dispositivo di puntamento (*button-mask* con valore 0 se il tasto è sollevato, valore 1 se il tasto è premuto). Per un mouse convenzionale, i bottoni 1, 2 e 3 corrispondono ai tasti sinistro, centrale e destro. Altri tipi di dispositivi sono supportati e gestiti per un massimo di 8 bottoni rappresentabili.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	5	*message-type*
1	U8		*button-mask*
2	U16		*x-position*
2	U16		*y-position*

Client CutText

Questo messaggio indica la presenza di caratteri codificati con ISO 8859-1 (Latin-1) nel buffer. La parte finale di una linea è rappresentata solamente dal carattere di nuova linea / (valore 10) senza la necessità di un valore di riporto (valore 13). Attualmente non c'è modo di trasferire testo diverso da quello rappresentato dalla codifica Latin-1.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	6	*message-type*
3			*padding*
4	U32		*length*
length	U8 array		*text*

2.3.4 Messaggi del Server

Questi messaggi sono inviati dal Server al Client a seguito di particolari richieste od a fronte di determinati eventi. Generalmente, i messaggi del Server riguardano l'aggiornamento del *Framebuffer* e tutte le varie specifiche grafiche relative al desktop remoto. Dal momento che il Protocollo *RFB*, come già stato detto, prevede la possibilità di essere esteso, il Server prima di inviare uno dei suoi messaggi al Client deve essersi assicurato della compatibilità delle regole tra Client e Server stabilite in fase di Inizializzazione. Di seguito sono riportati i messaggi predefiniti del protocollo inviati dal Server al Client.

Aggiornamento del Framebuffer

L'aggiornamento del *Framebuffer* (*FramebufferUpdate*) consiste in una sequenza di rettangoli di dati pixel che il Server invia al Client, il quale provvederà ad inserirli nel suo *Framebuffer* locale ed aggiornare il suo stato. Questi messaggi sono inviati al Client in risposta ad una richiesta di aggiornamento del *Framebuffer* (*FramebufferRequest*, sezione 2.3.3)

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	0	*message-type*
1			*padding*
2	U16		*number-of-rectangles*

Segue poi una serie di rettangoli di dati pixel quantificata dal valore *number-of-rectangles*. Ogni rettangolo è così rappresentato:

Num.Byte	Tipo	(Valore)	Descrizione
2	U16	0	*x-position*
2	U16		*y-position*
2	U16		*width*
2	U16		*height*
4	S32		*encoding-type*

Infine, se specificato, segue il tipo di codifica da applicare nella rappresentazione dei dati pixel. Per un elenco completo di tali codifiche si rimanda alla sezione 2.3.5.

Mappatura dei colori

Quando il formato dei pixel richiede l'utilizzo di una mappa di colori, questo messaggio informa il Client che gli specifici valori dei pixel devono essere determinati da una fissata mappatura di intensità RGB.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	1	*message-type*
1			*padding*
2	U16		*first-colour*
2	U16		*number-of-colours*

Il messaggio è seguito da una sequenza di numeri di colore (*number-of-colours*) così composta:

Num.Byte	Tipo	(Valore)	Descrizione
2	U16		*red*
2	U16		*green*
2	U16		*blue*

Allarme

In caso di necessità, fine ultimo di questo messaggio è di suonare un allarme acustico sul lato Client, se presente.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	2	*message-type*

Server CutText

Questo messaggio indica la presenza di caratteri codificati con ISO 8859-1 (Latin-1) nel buffer. La parte finale di una linea è rappresentata solamente dal carattere di nuova linea / (valore 10) senza la necessità di un valore di riporto (valore 13). Attualmente non c'è modo di trasferire testo diverso da quello rappresentato dalla codifica Latin-1.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8	6	*message-type*
3			*padding*
4	U32		*length*
length	U8 array		*text*

2.3.5 Codifiche

Come già accennato in precedenza, le codifiche dei valori pixel indicano il modo con cui deve essere rappresentato un rettangolo. Esistono diversi tipi di codifiche utilizzabili a seconda delle esigenze del Client, della sua rapidità di processare i dati ricevuti o della velocità della rete.

Segue un elenco dei principali tipi di codifica definiti nel Protocollo *RFB*.

Codifica Raw

Si tratta della codifica più semplice, in cui ogni dato consiste nella dimensione in termini di *larghezza* × *altezza* (*width* × *height*) di pixel del rettangolo da rappresentare in uno specifico ordine da sinistra a destra. Ogni Client supporta questo tipo di codifica che viene eseguita dai Server a meno di una richiesta diversa da parte del Client.

Num.Byte	Tipo (Valore)	Descrizione
$width \times height \times bytesPerPixel$	PIXEL array	*pixels*

Codifica Copy Rectangle

È un tipo di codifica semplice ed efficiente che consiste nell'inviare una coppia di coordinate X, Y rappresentanti la posizione sullo schermo in cui copiare un fissato rettangolo di dati pixel memorizzati all'interno del *Framebuffer*. Tale codifica risulta particolarmente efficiente in molti casi di semplice utilizzo dell'interfaccia grafica da parte dell'utente, come ad esempio lo spostamento di una finestra attraverso lo schermo o quando il contenuto di una finestra viene scrollato.

Num.Byte	Tipo (Valore)	Descrizione
2	U16	*src-x-position*
2	U16	*src-y-position*

Codifica RRE

Nella codifica RRE (*Rise-and-Run-Length*) ogni rettangolo non solo viene compresso con un alto grado di ottimizzazione, ma arriva al Client in una forma con cui può essere rappresentato nel modo più immediato ed efficiente possibile. Ciò avviene partizionando l'intera l'area del rettangolo in tanti piccoli sottorettangoli ognuno dei quali consiste in un singolo valore pixel. Tali sottorettangoli, una volta riuniti, formeranno nuovamente la regione rettangolare di partenza. Il risultato della codifica inviato sulla rete contiene il valore del colore dei pixel di background V_b, ed un valore numerico N, seguiti da una lista di N sottorettangoli ognuno dei quali è formato da una quintupla contenente il valore (v) del pixel, le coordinate (x,y) del sottorettangolo relativo all'angolo superiore sinistro della regione principale ed i valori di larghezza ed altezza (w,h) del sottorettangolo.

Il messaggio contenente il valore di questa codifica inizia con un header:

Num.Byte	Tipo (Valore)	Descrizione
4	U32	*number-of-subrectangles*
bytesPerPixel	PIXEL	*background-pixel-value*

Segue poi un elenco contenete tutti i sottorettangoli richiesti. La lunghezza di tale elenco è specificata dal valore *number-of-subrectangles*. Ogni sottorettangolo possiede le seguenti informazioni:

Num.Byte	Tipo (Valore)	Descrizione
bytesPerPixel	PIXEL	*subrect-pixel-value*
2	U16	*x-position*
2	U16	*y-position*
2	U16	*width*
2	U16	*height*

Codifica Hextile

La codifica Hextile rappresenta una variante della codifica *RRE*: ogni regione rettangolare viene partizionata in piccoli quadratini detti *tiles* da 16x16 pixel formati da 4 bit ciascuno per un totale di 16 bit. La suddivisione del rettangolo originario viene eseguita in un modo predeterminato in modo tale da evitare di specificare la posizione degli stessi per poter creare nuovamente la regione rettangolare: ogni *tile* viene ordinato al proprio posto a partire dalla posizione dell'angolo superiore sinistro fino ad arrivare all'angolo inferiore destro, generando il rettangolo con un ordinamento che va da sinistra a destra e dall'alto al basso. Se la larghezza o l'altezza del rettangolo non sono multipli esatti di 16, gli ultimi *tile* della partizione avranno una dimensione inferiore a quella specificata inizialmente. Ogni *tile* è codificato con la la codifica *Raw* o *RRE* e può possedere il valore del colore di background e foreground: tali valori possono essere omessi se equivalgono a quelli dei *tiles* precedenti; in tali casi si fa riferimento al valore del *tile* che viene prima.

Ogni *tile* è rappresentato da una sua specifica codifica che indica il modo con cui determinare l'intensità ed il valore del colore di backgorund e foreground:

Num.Byte	Tipo	(Valore)	Descrizione
bytesPerPixel	PIXEL		*subrect-pixel-value*
1	U8		*subencoding-mask*
		1	*Raw*
		2	*BackgroundSpecified*
		4	*ForegroundSpecified*
		8	*AnySubrects*
		16	*SubrectsColoured*

Se il bit *Raw* è impostato con un valore diverso da zero, tutti gli altri bit sono privi di ogni rilevanza e seguono le dimensioni di ogni *tile* in termini di larghezza (*width*) ed altezza (*height*). Altrimenti i successivi bit sono specificati come segue:

BackgroundSpecified specifica il colore di background del *tile* cui fa riferimento. Tale valore deve essere necessariamente impostato per il primo *tile* del sottorettangolo con una codifica diversa da *Raw*. Se questo bit non viene impostato, allora si utilizza il valore del colore di background utilizzato dall'ultimo *tile* processato.

Num.Byte	Tipo	(Valore)	Descrizione
bytesPerPixel	PIXEL		*background-pixel-value*

ForegroundSpecified rappresenta il colore di foreground del *tile* cui fa riferimento. Se questo bit viene impostato, allora *SubrectsColoured* deve necessariamente avere valore zero.

Num.Byte	Tipo	(Valore)	Descrizione
bytesPerPixel	PIXEL		*foreground-pixel-value*

AnySubrects indica il numero di sottorettangoli che seguono il messaggio ed il Client dovrà ricevere. Se tale bit non sarà impostato con nessun valore, allora non seguiranno sottorettangoli.

Num.Byte	Tipo	(Valore)	Descrizione
1	U8		*number-of-subrectangles*

SubrectsColoured specifica il colore proprio di ogni sottorettangolo. In tal modo, i sottorettangoli che seguono il messaggio rispettano la struttura seguente:

Num.Byte	Tipo (Valore)	Descrizione
bytesPerPixel	PIXEL	*subrects-pixel-value*
1	U8	*x-and-y-position*
1	U8	*width-and-height*

Se invece *SubrectsColoured* non è impostato con nessun tipo di valore, tutti i sottorettangoli che seguono devono essere rappresentati con lo stesso colore di foreground (*ForegroundSpecified*) o, in alternativa, dell'ultimo *tile* rappresentato. In tal caso, la struttura dei sottorettangoli sarà così formata:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*x-and-y-position*
1	U8	*width-and-height*

Codifica ZRLE

ZRLE è acronimo di *Zlib ([7]) Run-Length Encoding*, ed è uno speciale tipo di codifica che combina la compressione Zlib, l'utilizzo di una tavola di colori, la suddivisione in quadratini e la codifica *Run-Length*. I rettangoli sono inviati per mezzo di una struttura formata da un campo di 4 bytes che indica la lunghezza del resto del messaggio, ed un certo numero di dati compressi per mezzo della compressione Zlib.

Num.Byte	Tipo (Valore)	Descrizione
4	U32	*length*
length	U8 array	*zlibData*

L'oggetto *zlibData*, una volta decompresso, rappresenta *tiles* di 64x64 pixel analoghi a quelli della codifica *Hextile* (sezione 2.3.5) La codifica ZRLE fa uso del nuovo tipo CPIXEL, uno speciale tipo di pixel compresso il cui formato è simile a quello PIXEL ad eccezione di *true-colour-flag* impostato a true, *bits-per-pixel* impostato a 32, *depth* con un valore non superiore a 24 e tutti i bit relativi alle intensità dei colori rosso, verde e blu formati dai byte più o meno significativi a seconda della situazione.

Ogni *tile* ha inizio con un byte che rappresenta il tipo di subcodifica, i cui possibili valori sono di seguito rappresentati:

0 **-** Il tipo di codifica del pixel utilizzata è la codifica *Raw*; seguono le dimensioni ($width \times height$) del sottorettangolo.

Num.Byte	Tipo (Valore)	Descrizione
width × *height* × *bytesPerCPixel*	CPIXEL array	*pixels*

1 - Un *tile* solido che consiste in un singolo colore fissato. Segue il valore del pixel.

Num.Byte	Tipo (Valore)	Descrizione
bytesPerCPixel	CPIXEL	*pixelValue*

2 a 16 - Viene utilizzato uno specifico tipo di tavola dei colori per determinare l'intensità dei valori dei pixel.

Num.Byte	Tipo (Valore)	Descrizione
paletteSize × *bytesPerCPixel*	CPIXEL array	*palette*
m	U8 array	*packedPixels*

17 a 127 - Inutilizzati.

128 - Viene pianificata la codifica di tipo RLE (*Run-Length Encoding*) che consiste in un certo numero di esecuzioni ripetute fino a quando il *tile* è completamente formato. Ogni esecuzione è rappresentata da un singolo valore pixel seguito dalla lunghezza di uno o più byte dell'esecuzione.

Num.Byte	Tipo (Valore)	Descrizione
bytesPerCPixel	CPIXEL	*pixelValue*
floor((runLength - 1)/255)	U8 array 255	
1	U8	*(runLength - 1)%255)*

129 - Inutilizzato.

130 a 255 - Viene pianificata la codifica di tipo RLE (*Run-Length Encoding*) insieme all'applicazione di una tavola di colori predefinita la cui dimensione è fissata: *paletteSize = (subencoding - 128)* pixel.

Num.Byte	Tipo (Valore)	Descrizione
paletteSize × *bytesPerCPixel*	CPIXEL array	*palette*

Segue poi un certo numero di esecuzioni ripetute, come da codifica RLE, la prima delle quali è rappresentata da un indice della tavola di colori:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*paletteIndex*

Le successive ripetizioni sono rappresentate dall'indice della tavola dei colori con il primo bit impostato, seguite dalla lunghezza dell'esecuzione:

Num.Byte	Tipo (Valore)	Descrizione
1	U8	*paletteIndex + 128*
floor((runLength - 1)/255)	U8 array 255	
1	U8	*(runLength - 1)%255*

2.3.6 Pseudo Codifiche

Questi particolari tipi di codifiche rappresentano vere e proprie estensioni al Protocollo *RFB*. In fase di Inizializzazione, Server e Client si accorderanno sui tipi di codifica che sono in grado di gestire e, quindi, utilizzare. Tra di esse vi sono anche le pseudo codifiche.

Di seguito sono riportate le due principali pseudo codifiche definite nel protocollo.

Cursore

Un Client che richiede una pseudo codifica del cursore dichiara di essere in grado di gestire localmente il tracciamento del cursore di un mouse. Tale pseudo codifica può notevolmente migliorare le prestazioni per tipi di collegamenti particolarmente lenti. Il Server imposta la forma del cursore inviando al Client uno pseudo rettangolo contenente la pseudo codifica del cursore come parte di un aggiornamento del *Framebuffer*. La posizione dello pseudo

rettangolo sullo schermo indica la posizione del cursore, mentre le dimensioni in pixel si riferiscono a quelle del cursore.

Num.Byte	Tipo	(Valore)	Descrizione
$width \times height \times bytesPerPixel$	PIXEL array	255	*cursor-pixels*
*floor((width + 7)/8) * height*	U8 array		*bitmask*

Dimensione dello schermo

Un Client che richiede una pseudo codifica della dimensione dello schermo (*DesktopSize*) dichiara di essere in grado di gestire un cambiamento della dimensione del *Framebuffer*. Il Server cambia la dimensione dello schermo inviando uno pseudo rettangolo come ultimo rettangolo di un aggiornamento del *Framebuffer* per mezzo di questa pseudo codifica. La posizione di tale pseudo rettangolo è ignorata, mentre i valori di dimensione *width* ed *height* rappresentano la nuova dimensione del *Framebuffer*.

Capitolo 3

Esperimento

Lo studio di un progetto che ha come fine l'analisi di un software VNC, in particolar modo un Client, da applicare nell'ambiente virtuale del web, coinvolge necessariamente un insieme di strumenti e tecnologie che riguardano dettagliatamente ogni ambito interessato. In questo capitolo verranno introdotti e presentati tali strumenti. Farà seguito una spiegazione del loro effettivo utilizzo, oltre ad un'illustrazione delle modalità con cui essi comunicano tra di loro in modo omogeneo ed efficace per raggiungere il risultato prefissato oggetto di questo studio.

3.1 Bochs e Windows 1.01

Come anticipato nel capitolo 1, l'importanza assunta dalle Macchine Virtuali nel panorama attuale raggiunge un livello considerevole. A tal proposito, l'utilizzo della Macchina Virtuale *Bochs* è parte integrante dello studio di questo progetto. Nella sezione 3.1 del capitolo sopracitato, è stato introdotto questo tipo di Macchina Virtuale come un emulatore di architetture x86 capace di eseguire diversi sistemi operativi su altrettante differenti workstation.

Bochs interagisce fortemente con il sistema operativo della piattaforma su cui è installato: la pressione di un tasto o di un bottone nel display di *Bochs* causa un evento che viene intercettato e gestito dal sistema virtuale; quando il sistema operativo emulato deve leggere dall'hard disk virtuale creato dalla Macchina, *Bochs* attinge tali dati da un file immagine situato sul computer ospitante; se si riscontra la necessità di inviare un pacchetto di dati sulla rete locale, *Bochs* si appoggia ad una speciale scheda di rete tramite cui avrà la facoltà di incapsulare i propri dati ed inviarli all'esterno del proprio sistema virtualizzato. Tutte queste interazioni con il sistema ospitante, per quanto

complicate e specifiche, offrono all'emulatore *Bochs* una notevole capacità di simulazione con performance ottimali che interessano la maggior parte dei sistemi operativi ed hardware attualmente esistenti.

L'utilizzo della Macchina Virtuale *Bochs* ha ricoperto un ruolo importante nello studio dei sistemi operativi, sia a titolo accademico che per la ricerca e lo sviluppo tecnologico ed informatico orientato al futuro. Ma non solo. Esso rappresenta un ottimo strumento che permette di creare una finestra sul passato, grazie alla possibilità di eseguire sistemi operativi datati ed obsoleti ma che rappresentano importanti capitoli nella storia dell'informatica e dei sistemi operativi: spesso, questi sistemi antiquati sono incompatibili sulle nuove macchine dotate di hardware impossibilitati ad eseguirli. Di conseguenza, a meno che non si abbia in dotazione macchinari originari del passato, assembrati per ospitare ed eseguire esattamente questi sistemi, è necessario appoggiarsi e far uso delle Macchine Virtuali, tra cui *Bochs*.

Per poter essere eseguito, *Bochs* richiede la presenza di pochi requisiti necessari ed indispensabili per il suo corretto funzionamento:

- L'eseguibile *Bochs.exe*: si tratta del programma vero e proprio, compilato e pronto per essere avviato sul sistema operativo ospitante. Per poter essere eseguito correttamente, è necessario configurarlo per mezzo di un particolare file di configurazione, solitamente nominato *bochsrc.txt*, che contiene un elenco completo di tutte le impostazioni personalizzabili del simulatore oltre ai percorsi locali tramite cui sarà possibile accedere ai vari file su cui *Bochs* basa e fonda la propria esecuzione. Per i dettagli del file di configurazione si rimanda alla consultazione della sezione 3.1.1;

- Immagini *BIOS* (*Basic Input-Output System*): questi file, rappresentati da due particolari tipi di immagine, *BIOS* e *VGA BIOS*, forniscono alla Macchina Virtuale un insieme di procedure che permettono di gestire il corretto funzionamento dell'accesso dell'hardware virtuale e di tutte le componenti che formano l'emulatore quali le periferiche ed i programmi software;

- Immagine del disco: è il vero e proprio disco virtuale che *Bochs* deve eseguire ed emulare sul proprio simulatore. Questa immagine è caricata all'avvio del sistema e può rappresentare un floppy disk, un CD-ROM, un hard-disk o addirittura un intero sistema operativo che, in fase di *boot*, *Bochs* si premunirà di avviare secondo la corretta procedura. Alcuni file di questo genere sono reperibili gratuitamente dal sito web

di *Bochs* ([8]), tra cui l'immagine del sistema operativo *Windows 1.01* (vedere la sezione 3.1.2).

3.1.1 Configurazione

Bochs utilizza un file di configurazione per conoscere esattamente il luogo in cui cercare i file immagine da caricare ed il comportamento che deve assumere sia nei confronti del sistema ospitante che in quelli delle componenti da lui virtualizzate al fine di garantire il corretto funzionamento del sistema di emulazione. Il file di configurazione, solitamente noto come *bochsrc.txt*, può essere modificato dall'utente a seconda delle sue necessità. In fase di avvio, la Macchina Virtuale legge il contenuto di questo file e lo analizza per poter essere in grado di applicarlo, quando necessario, durante il funzionamento e l'esecuzione.

Di seguito è riportato un elenco con le principali opzioni che è possibile configurare all'interno del file:

- **plugin_ctrl**: controlla la presenza di eventuali dispositivi di plugin che vengono caricati all'avvio dell'emulatore a seconda dell'impostazione qui configurata: il valore "1" indica che il plugin deve essere caricato; il valore "0", al contrario, indica che il plugin deve essere disattivato se precedentemente caricato. Alcuni plugin sono caricati di default, mentre per altri è necessario specificare nel file di configurazione la richiesta di caricamento;

- **memory**: imposta il valore totale di memoria che si desidera virtualizzare durante l'emulazione. È possibile distinguere sia la memoria destinata all'utente, che la memoria RAM. La memoria allocata virtualmente dovrà essere necessariamente inferiore a quella disponibile nel sistema operativo ospitante;

- **cpu**: definisce i parametri del processore virtuale utilizzato da *Bochs*. Tali parametri riguardano, ad esempio, il numero massimo di processori utilizzabili per un singolo thread (da 1 fino ad un massimo di 8); il numero massimo di istruzioni che un singolo processore è in grado di eseguire, prima di passare il controllo ad un altro processore; il numero massimo di *IPS* (*Instructions Per Second*) che *Bochs* sarà in grado di eseguire sulla macchina virtuale; il reset automatico della CPU in caso si verifichino tre criticità consecutive;

- **romimage**: specifica il pathname dell'immagine ROM eseguita in fase di avvio della Macchina Virtuale e che ne controlla il corretto funzionamento. Di norma viene utilizzato un *BIOS* precompilato situato nella stessa directory che contiene i file sorgente ed i file binari di *Bochs*. È possibile inoltre utilizzare un *BIOS* diverso da quello usato di default, specificandone l'indirizzo all'interno del file di configurazione;

- **optromimage**: questa opzione consente di caricare ed utilizzare fino a quattro immagini ROM opzionali. In questo caso è necessario assicurarsi di utilizzare aree di memoria da cui è permessa soltanto la lettura. Le immagini ROM opzionali non devono in alcun modo sovrascrivere la memoria utilizzata dai file *BIOS* della Macchina Virtuale;

- **vgaromimage**: tale parametro specifica il pathname della *VGA ROM BIOS* che deve essere utilizzata da *Bochs*;

- **vga**: questa opzione definisce i parametri relativi al display *VGA* (*Video Graphics Array*), uno standard grafico che codifica e gestisce un flusso di dati visualizzati graficamente su un display. È possibile modificare l'estensione del display e la frequenza di aggiornamento dell'immagine basata sul clock dell'emulatore *Bochs*;

- **floppy**: permette a *Bochs* di accedere ed utilizzare fino a due floppy disk presenti sul computer che lo ospita. È inoltre possibile l'utilizzo di file immagine che rappresentano e virtualizzano i dispositivi floppy specificando in tale parametro di configurazione il pathname dell'immagine floppy alla quale attingere;

- **boot**: definisce la sequenza di *boot*. È possibile, in fase di avviamento, specificare fino a tre dispositivi tra "floppy", "disk", "cdrom" o "network";

- **config_interface**: consiste in una serie di menu e finestre di dialogo da cui è possibile modificare i parametri di configurazione di *Bochs*. In base alla piattaforma che si utilizza, sono possibili tre diverse scelte dell'interfaccia di configurazione: una versione testuale, una versione specifica per i sistemi Windows a 32 bit, una versione utilizzabile soltanto nei casi in cui *Bochs* è compilato per mezzo del supporto wxWidgets;

- **display_library**: codifica la modalità di visualizzazione dello schermo *VGA* della Macchina Virtuale. Esistono ben dieci diverse implementazioni con cui è possibile visualizzare il display sulle varie piattaforme, a seconda delle proprie necessità e dell'uso che se ne intende fare. Ci

sono quindi librerie che gestiscono la visualizzazione su sistemi Mac, Windows, Linux e per le connessioni che fanno uso del Protocollo *RFB*;

- **log**: definisce il pathname del file testuale di log che *Bochs* crea automaticamente per tener traccia delle varie operazioni compiute, eventuali errori che si presentano durante l'esecuzione del sistema, etc;
- **keyboard**: definisce i parametri relativi all'emulazione della tastiera, in modo tale da specificarne il tipo, la lingua, il ritardo che intercorre tra la pressione di un tasto ed il trasferimento del dato, la mappatura virtuale di tutti i caratteri rappresentati e rappresentabili e la definizione degli shortcut ammessi;
- **mouse**: definisce i parametri relativi all'emulazione del mouse; si può selezionare ed impostare un determinato tipo di mouse, gestire gli eventi che si generano alla pressione dei bottoni del dispositivo o addirittura disabilitarlo.

Per un elenco completo e dettagliato di tutte le opzioni di configurazione del file *bochsrc.txt*, si rimanda alla consultazione della documentazione ufficiale presente sul sito di *Bochs* [8].

3.1.2 Windows 1.01

Il sistema operativo *Windows 1.01* rappresenta la prima versione funzionante ed operativa del famoso sistema operativo di casa Microsoft, rilasciato nel 1985, che implementa un ambiente operativo multitasking basato su una funzionale e semplice interfaccia grafica. Questo sistema operativo includeva driver originali per le diverse componenti con cui era formato, e le sue applicazioni potevano richiamare soltanto le *API* basate su questi driver, accedendo quindi direttamente all'hardware od al BIOS per poter svolgere il proprio lavoro. Tale versione del sistema operativo Windows eseguiva la shell MS-DOS ed era dotata di programmi quali Calcolatrice, Calendario, Orologio, Blocco Note, Pannello di Controllo, Prompt dei comandi e vari altri.

Il sistema operativo *Windows 1.01* si rivela quindi particolarmente adatto per uno studio della struttura e dell'architettura dei sistemi operativi, oltre a fornire un tassello importante che testimonia la storia dello sviluppo dei sistemi, dalle origini fino all'attualità.

Sul sito web ufficiale della Macchina Virtuale *Bochs* [8], è disponibile e scaricabile un file immagine contenente il sistema operativo *Windows 1.01*.

Tale file, se passato all'emulatore *Bochs* installato su un qualsiasi sistema operativo, permette di avviare *Windows 1.01* in un ambiente completamente virtualizzato ed interattivo.

Nel file di configurazione, l'opzione che specifica l'utilizzo del file immagine con il sistema operativo *Windows 1.01*, nominato *win101.img*, è la seguente:

```
ata0-master: type=disk, path="$BXSHARE\Windows 1.01\win101.img"
```

Dove:

- *ata0-master*: è l'opzione che definisce i tipi e le caratteristiche di tutti gli eventuali dispositivi che possono far parte del sistema virtualizzato;
- *type*: è il parametro che rappresenta il tipo del dispositivo che può essere "disk" o "cdrom";
- *path*: è il parametro che indica il percorso completo all'interno del file system ove bisogna cercare il dispositivo richiesto dall'opzione.

3.2 HTML5 e GWT

La grande espansione della tecnologia informatica, in particolare lo sviluppo nell'ambito dell'intercomunicazione tra dispositivi fissi e mobili, computer e tablet, smartphone e console, sta avendo un sempre maggior rilievo. Assume quindi un ruolo di primaria Importanza l'esistenza di una rete globale raggiungibile ovunque, ed il cui accesso ed utilizzo non deve essere limitato dagli strumenti a propria disposizione. La pluralità dei dispositivi hardware e software in circolazione devono essere in grado di interfacciarsi tra di loro e con la rete globale, attingere pacchetti da essa e trasmettere dati nel modo più efficace ed efficiente possibile.

Per questi motivi, negli ultimi anni si è assistito alla nascita di speciali istituti, organizzazioni e consorzi con lo scopo di studiare e sviluppare tutte le potenzialità della rete mondiale di comunicazione, fino a progettare e creare un vero e proprio standard di linguaggio per la creazione di pagine web eseguibili su ogni tipo di software e dispositivo e capaci di gestire applicazioni e grandi flussi di dati oltre a fornire strumenti di sviluppo che possano garantire agli sviluppatori web un corretto utilizzo.

3.2.1 HTML5

Tali risultati sono stati raggiunti con la creazione di *HTML5*, un nuovo insieme di tecnologie orientate al web frutto del lavoro di due gruppi di lavoro: il *Web Hypertext Application Technology Working Group* (*WHATWG*, [3]), fondato da sviluppatori Apple, Mozilla Foundation ed Opera Software, ed il *World Wide Web Consortium* (*W3C*, [1]), un'organizzazione non governativa cui fan parte le principali aziende informatiche, della telefonia, università ed altre organizzazioni no-profit. Scopo ed obiettivo di questa ricerca era quello di progettare specifiche per le applicazioni web focalizzandosi nel miglioramento del linguaggio *HTML* e delle tecnologie ad esso correlate.

Evoluzione della quarta versione dello standard del linguaggio di markup *HTML*, è riduttivo definire a sua volta *HTML5* un linguaggio di markup, poiché le sue potenzialità sono di gran lunga superiori al suo predecessore. Ciò nonostante, una prima, grande innovazione rappresentata dallo standard *HTML5* consiste nel miglioramento della struttura della pagina web con l'introduzione di nuovi elementi di markup dotati di una propria semantica, caratteristica che è in grado di semplificare notevolmente tanto la creazione e l'accessibilità quanto l'analisi e lo studio di una pagina web. Questa innovazione comporta inoltre una migliore integrazione tra la struttura della pagina definita dai markup, le caratteristiche di resa definite dalle direttive di stile, ed i contenuti della pagina stessa.

Ulteriore scopo del nuovo standard *HTML5* consiste nella trasformazione ideale di Internet da una immensa biblioteca di pagine e documenti consultabili, ad una sconfinata raccolta di applicazioni web, tale da rendere Internet un vero e proprio archivio a cui attingere ogni qualvolta se ne presenta l'occasione. Di conseguenza, la gestione delle applicazioni web e delle pagine strettamente correlate ad esse rappresenta un altro importante tassello che caratterizza *HTML5*. Queste applicazioni spesso devono gestire un ingente e notevole flusso di informazioni, ed *HTML5* migliora e valorizza il metodo di immagazzinamento e salvataggio dei dati, fornendo ai browser la capacità di selezionare un certo numero di file da salvare in locale per poter consentire la consultazione di una pagina e l'utilizzo di un'applicazione anche offline in assenza di connessione, nonché un'evoluzione dei cookies con una migliore gestione degli stessi ed un ampliamento della memoria a loro disposizione, così da salvare i dati dal browser a cui attingere senza dover effettuare una connessione al database centralizzato.

HTML5 è uno standard orientato al futuro. Tra le varie innovazioni che

introduce, molte, non meno importanti delle precedenti, riguardano la gestione e lo sfruttamento delle potenzialità di ogni genere di dispositivo in grado di effettuare una connessione ad Internet e, di conseguenza, poter usufruire di *HTML5*. Un nutrito numero di *API* (*Application Programming Interface*), particolari procedure e librerie disponibili per i linguaggi di programmazione, permettono di far uso delle caratteristiche peculiari e specifiche di ogni dispositivo, come ad esempio la fotocamera, la rubrica od il microfono, oltre a fornire un importante sistema di geolocalizzazione per mezzo del quale è possibile accedere direttamente a tutte le informazioni geografiche sul luogo in cui un dispositivo si trova e tramite esse offrire all'utente un certo tipo di servizi altrimenti impossibili da garantire.

Infine, *HTML5* non è estraneo alla direzione intrapresa dalle moderne tecnologie, fortemente indirizzate ad un uso massiccio di elementi multimediali di elevata qualità. Per far fronte a questo andamento, *HTML5* mette a disposizione tag ed elementi che non solo consentono di gestire e rappresentare in maniera pratica e veloce, senza l'ausilio di plugin esterni, notevoli flussi di dati multimediali, come filmati ed audio, ma, nel caso di *Canvas*, permettono di modificare e manipolare in tempo reale video ed immagini offrendo un comodo supporto per animazioni dinamiche sia 2D che 3D.

Dopo questa panoramica, si nota come *HTML5* sia uno standard ideale per la gestione di tutto ciò che compete il web, fortemente orientato al futuro e che offre a tutti gli utenti l'opportunità di beneficiare della sua tecnologia. Tale standard è ancora in fase di elaborazione, per quanto buona parte di esso sia attualmente applicato ed utilizzato nel web, e si presume che una prima versione ufficiale verrà pubblicata non prima della fine del 2014.

3.2.2 GWT

GWT (*Google Web Toolkit*) è un insieme di strumenti per lo sviluppo, la creazione e l'ottimizzazione di applicazioni web. Offerto gratuitamente da Google con lo scopo di promuovere lo sviluppo di software ottimizzato per il web anche da parte di sviluppatori non esperti delle dinamiche dei browser, di JavaScript e delle tecniche di comunicazione del protocollo Http, *GWT* permette la creazione di pagine web *HTML5* e pagine dinamiche che possono contenere al loro interno applicazioni, in modo trasparente e perfettamente compatibile con qualsiasi tipo di browser web utilizzato.

Il kit di sviluppo del software (*SDK*, *Software Development Kit*) di *GWT* fornisce un insieme di procedure e librerie *API* che permettono di scrivere

applicazioni *AJAX* (*Asyncronus JavaScript And XML*) in Java e compilare il codice in JavaScript ottimizzato per essere eseguito su qualsiasi browser, incluse le versioni mobile. Le applicazioni web *AJAX* sono applicazioni interattive e permettono uno scambio di dati asincrono tra un web browser ed un Server in background senza interferire con il comportamento della pagina web dinamica che viene quindi costantemente aggiornata in modo automatizzato e continuo. La costruzione di applicazioni *AJAX* per mezzo di *GWT* garantisce un alto livello di astrazione che permette una maggior produttività senza l'obbligo di manipolare direttamente il modello del documento della pagina web e le richieste di connessione del protocollo Http.

GWT può essere utilizzato sia per mezzo di un *IDE* (*Integrated Development Environment*), un particolare ambiente di lavoro integrato che permette la creazione e lo sviluppo di applicazioni software, oppure direttamente in un browser web grazie ad un particolare plugin, il *GWT Developer Plugin*. Questo permette ampia libertà nell'utilizzo di tali strumenti, in particolar modo nella fase di scrittura e controllo del codice, della sua ottimizzazione e dell'esecuzione per verificarne il funzionamento. Di conseguenza è possibile eseguire la fase di Debug, sia nell'*IDE* utilizzato nella scrittura del codice, ed in tal caso le applicazioni *AJAX* assumono lo stesso comportamento che avrebbero nel caso di una vera e propria applicazione desktop, sia direttamente nel browser web in cui il plugin occupa il divario tra il codice JavaScript letto dal browser ed il bytecode Java dell'applicazione senza il bisogno di doverlo compilare ogni volta che viene apportata una modifica: sarà sufficiente aggiornare manualmente la pagina web affinché siano applicate le modifiche apportate al codice dell'applicazione. Una prima, importante ottimizzazione delle applicazioni web viene eseguita da *GWT* che è in grado di apportare modifiche al sorgente quali la rimozione di codice o variabili inutilizzate, l'ottimizzazione di stringhe e la segmentazione del codice e dei metodi in modo tale da rendere veloce ed efficiente il download e l'utilizzo dell'applicazione sviluppata. Infine, *GWT* offre la possibilità di eseguire la propria applicazione web compilando il codice Java in file JavaScript compatibili con tutti i principali browser web esistenti, compresi quelli mobile per smartphone, e pubblicarla su uno store così da renderla disponibile a chiunque desideri scaricarla e voglia usufruire di tale servizio.

3.3 Client VNC

Attualmente esistono una pluralità di programmi, sia offerti gratuitamente che disponibili a pagamento, che permettono di stabilire una connessione

virtuale e remota tra due computer. Questi tipi di software *VNC*, come già spiegato nella sezione 1.3.1 del capitolo 1, oltre a consentire il collegamento tra due dispositivi, differenziandosi per tipo di piattaforma e sistema operativo, offrono altri tipi di servizi come il trasferimento dei dati, lo scambio di messaggi in tempo reale stile chat, canali di trasmissione dedicati, alti livelli di sicurezza e codifica dei dati.

Tali software *VNC* sono reperibili sui siti web delle case di produzione, dai quali si può effettuare il download dell'esecutivo, Client o Server, da installare sul proprio computer o dispositivo. Attraverso questo software installato sarà possibile richiedere il collegamento ad un Server e stabilirvi la connessione remota. Per quanto questa soluzione sia comoda ed efficiente, molto spesso ci si trova nella condizione di non poter installare sul computer sul quale si opera un software aggiuntivo, applicazione o plugin che permette all'utente la connessione remota. Tale limitazione può rappresentare un problema non indifferente, che richiede quindi la presenza di ulteriori soluzioni.

Una valida alternativa è rappresentata da *WebVNC* [2], un Client *VNC* creato da *Alexandr de Pellegrin* e *Victor Freches*. Sviluppato con il kit *GWT*, fa uso della tecnologia *AJAX* per stabilire una connessione remota tra un Client ed un Server, tramite web, senza dover scaricare ed installare sul proprio dispositivo nessun tipo di software, applicazione o plugin. Lo sviluppo di un Client *VNC* con gli strumenti di *GWT* permette un ampio utilizzo del Client ed una sua maggior portabilità su gran parte dei sistemi operativi, hardware e software. In particolar modo, tramite *WebVNC* si può accedere da un qualsiasi computer che abbia accesso ad internet, in ogni parte del mondo, ad una pagina web sviluppata con *HTML5*. Da questa pagina, inserendo i dati opportuni, sarà possibile stabilire una connessione remota con un qualsiasi Computer Server senza dover installare nessun tipo di programma sul Computer Client: è sufficiente connettersi alla pagina web e fornire i dati di connessione ed autenticazione se richiesta.

Il progetto *WebVNC* implementa tutte le specifiche del protocollo *RFB*, ma oltre a questo si pone l'obiettivo di fornire una quanto più ampia generalizzazione nel metodo di comunicazione grafica tra due computer, un Client ed un Server, per mezzo di una connessione remota. Per questo motivo pone buona parte della propria attenzione anche allo studio ed allo sviluppo di un altro tipo di protocollo, il Protocollo *RDP* (sezione 1.3.2 del capitolo 1). Il programma implementato da *WebVNC*, oltre ad essere formato dalle componenti standard di un software *VNC* quali Client *VNC*, Server *VNC* ed un Protocollo di comunicazione (*RFB* e *RDP*), sviluppa un Server web aggiun-

tivo che ha il compito di ricevere ed inviare i dati tra le due parti: questo Server web, comunica con il Client *VNC* per mezzo di *WebSocket* fornite da *GWT*, speciali procedure che permettono una comunicazione bidirezionale attraverso una connessione TCP, e comunica con il Server *VNC* facendo uso del Protocollo di comunicazione (*RFB* o *RDP*).

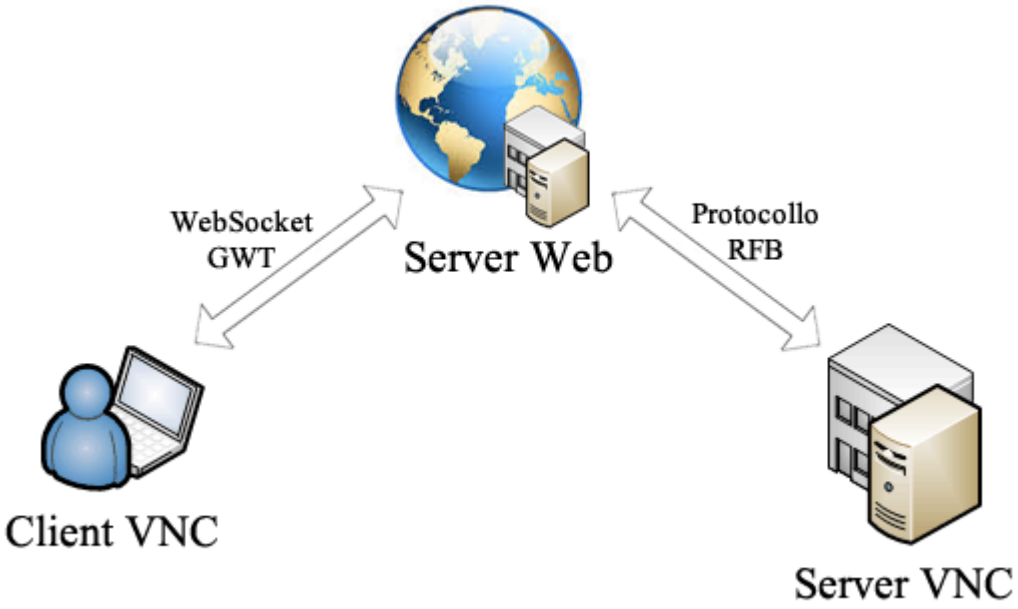

Figura 3.1: Sistema di comunicazione di WebVNC

3.3.1 Struttura del progetto

WebVNC rispetta la struttura standard dei progetti adottata da *GWT*, con cui è sviluppato. Il codice sorgente è contenuto nella directory *src* del package del progetto, all'interno della quale le unità di configurazione sono suddivise secondo una rigida gerarchia che definisce e raccoglie i moduli e le classi relativi alla programmazione del lato Client e quelli relativi al lato Server.

- **fr.neticar.webremoteclient.client**: questo package contiene tutti i moduli e le classi che gestiscono il lato Client.
  - *WRC.java*: è il modulo principale dell'applicativo del Client, che implementa l'*Entry Point*, il metodo eseguito all'avvio del programma. Contiene il codice relativo alla strutturazione ed alla

creazione della pagina web a cui l'utente dovrà connettersi per poter stabilire il collegamento con un Server. Inoltre raccoglie anche le chiamate delle funzioni che gestiscono l'avvio e la chiusura della connessione, nonché l'aggiornamento continuo delle immagini e dei pacchetti ricevuti dal Server;

- *WRCClientGatewayAsync.java*: è il modulo che contiene la classe che definisce le chiamate inviate al Server dal Client riguardanti l'avvio del Viewer, la sua chiusura, l'invio ed il ricevimento di eventi e dati;
- *LoginPanel.java*: questo modulo implementa la sezione della pagina web relativa al pannello di controllo contenente tutte le istruzioni che l'utente dovrà eseguire per interagire in modo costruttivo ed esauriente con la Macchina Virtuale remota;
- *ImagePanel.java*: questo modulo implementa la sezione della pagina web che visualizzerà in modo grafico i dati ricevuti dal Server durante la connessione. Gestisce gli eventi da tastiera e traccia il movimento del mouse codificandoli in informazioni da inviare al Server durante il collegamento.

- **fr.neticar.webremoteclient.server**: questo package contiene i moduli che implementano un Server *Jetty* che si pone nel mezzo della comunicazione tra Client *VNC* e Server *VNC*. Tale Server è il passo centrale dell'opera e prende parte attiva alla comunicazione, interagendo sia con il Server *VNC*, da cui preleva i dati desktop, sia con il Client *VNC*, che dovrà visualizzare le informazioni ricevute ed impacchettare altri dati da inviare, in risposta, al Server.

  - *WRCClientGatewayImpl.java*: questo modulo implementa il codice che si occupa di avviare una nuova sessione nel Server *Jetty*, grazie alla quale sarà possibile stabilire la connessione operativa tra Client *VNC* e Server *VNC*. Contiene inoltre le funzioni che gestiscono la ricezione e l'invio degli eventi, oltre a creare e configurare di tutti i file necessari per il collegamento;
  - *JettyServer.java*: questo modulo ha il compito di avviare il Server web che si pone in attesa delle richieste di connessione da parte del Client e farà da ponte nella comunicazione tra Client *VNC* e Server *VNC*, interagendo con entrambi al fine di espletare al meglio ogni operazione che da vita al collegamento remoto;
  - *VNCClient.java*: questo modulo contiene la definizione di tutte le funzioni che permettono la comunicazione tra Client e Server. In

particolar modo, gestisce il passaggio dei parametri e dei risultati delle diverse funzioni chiamate ed eseguite durante l'esecuzione dell'applicazione, mettendoli a disposizione ai moduli relativi al lato Client ed ai moduli relativi al lato Server.

- **fr.neticar.webremoteclient.server.vncviewer**: questo package contiene tutti i moduli e le classi che gestiscono il lato Server.
  - *RfbProto.java*: questo modulo implementa tutte le regole e le specifiche che definiscono il Protocollo *RFB* utilizzato da Client e Server per e nella connessione. Contiene quindi le variabili che definiscono la versione applicata, il tipo di codifica utilizzata per la trasmissione dei dati e dei pacchetti, nonché per la gestione della compressione delle immagini ricevute e degli eventi quali, ad esempio, il riconoscimento della tastiera e dei suoi caratteri ed il movimento del mouse con il tracciamento della posizione del puntatore. Le funzioni qui implementate rispettano fortemente la struttura del protocollo ufficiale *RFB* (per i dettagli si rimanda al capitolo 2);
  - *VncViewer.java*: questo modulo implementa al dettaglio tutte le classi che gestiscono la connessione e disconnessione da parte del Server, il ricevimento e l'invio dei pacchetti contenenti i dati necessari per stabilire il collegamento tra il lato Client ed il lato Server. Metodi come `init()`, `run()`, `connectAndAuthenticate()` e `doProtocolInitialization()` vengono utilizzati in fase di *Handshaking*, quella fase che precede lo stabilimento della connessione vera e propria e durante la quale Client e Server si scambiano tutte le informazioni necessarie affinché il collegamento si crei in modo stabile e tale da essere compatibile con entrambe le parti. Gli altri metodi che compongono questo modulo formano il corpo vero e proprio della connessione, durante la quale si scambiano i messaggi e si gestisce infine il termine e la chiusura della connessione;
  - *VncCanvas.java*: questo modulo raccoglie la parte del codice relativa alla renderizzazione delle immagini che il Client dovrà effettuare; per ogni pacchetto ricevuto, il Client deve applicare il formato e la codifica dei pixel utilizzati dal Protocollo *RFB* e decisi in fase di Handshaking. Una volta che sono stati stabiliti tali parametri, il modulo con opportuni metodi e classi provvede ad impacchettare i dati grafici ed inviarli continuamente al Viewer del Client, il quale si dovrà occupare della loro visualizzazione e rappresentazione nel riquadro Canvas che compone la pagina web.

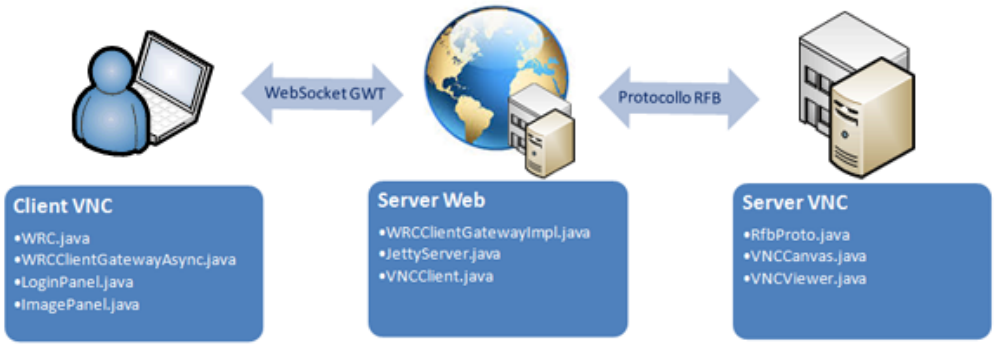

Figura 3.2: Struttura del progetto

3.3.2 Esecuzione

L'esecuzione del programma avviene con il lancio della classe *WRC* che implementa l'*Entry Point*, situato all'interno del modulo *WRC.java*. Il metodo `onModuleLoad()` è il primo ad essere eseguito, chiamato automaticamente all'avvio del programma, e come prima cosa genera la pagina web tramite cui l'utente potrà effettuare la connessione remota. Tale pagina è formata da due sezioni specifiche: la sezione che occupa la parte sinistra della pagina contiene un particolare pannello, *ImagePanel*, implementato dalla classe *ImagePanel.java*, che visualizzerà le informazioni ricevute durante la connessione; sarà in questo pannello che l'utente potrà interagire con i dati del Server. La parte destra della pagina web, invece, contiene un pannello verticale, *LoginPanel*, sviluppato dalla classe *LoginPanel.java*, all'interno del quale vi saranno alcune semplici istruzioni testuali che l'utente visitatore potrà seguire per interagire dinamicamente con la Macchina Virtuale, eseguendo operazioni in remoto sulla Macchina Virtuale stessa. È inoltre presente un bottone "Chiudi" tramite cui sarà possibile interrompere e terminare la connessione remota con il Server *VNC*.

Contemporaneamente alla creazione della pagina web da parte del Client, viene avviato anche il *Jetty Server*, il Server web che si frappone tra Client *VNC* e Server *VNC*. La sua creazione ed il suo avvio sono gestiti dal metodo `start()` della classe *JettyServer* presente nel relativo modulo.

Subito dopo aver creato la pagina ed aver avviato il Server web, il metodo `onModuleLoad()` alloca tre particolari variabili di tipo *String* impostandone il rispettivo valore. Queste tre variabili sono `host = "localhost"`,

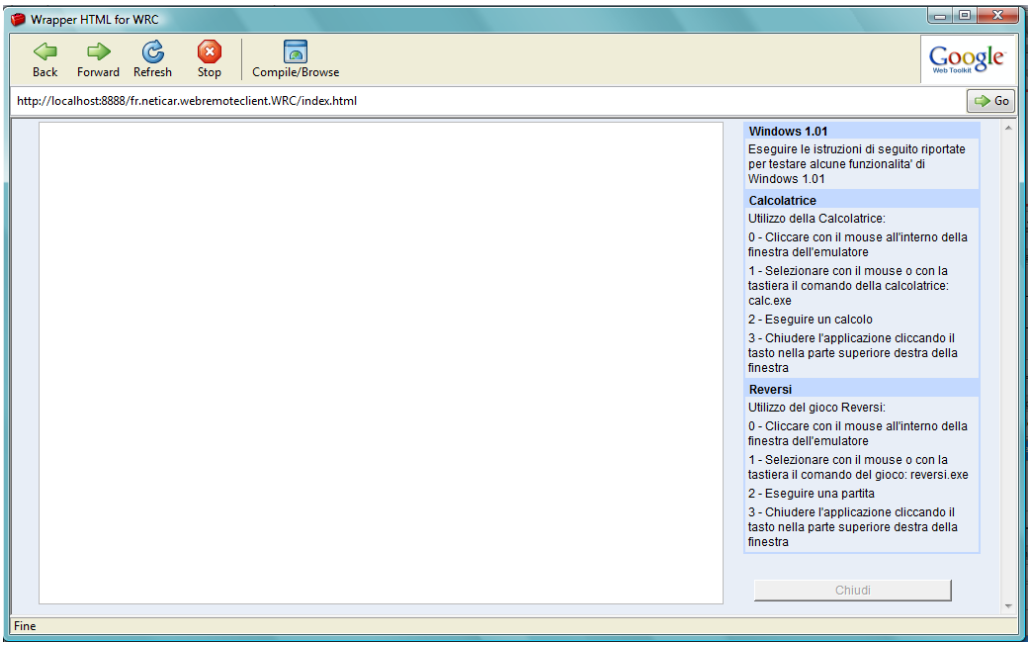

Figura 3.3: Struttura della pagina web

`port = "8888"` e `password = ""` corrispondono all'indirizzo ed alla porta del Server web al quale il Client *VNC* dovrà connettersi ed alla password da utilizzare se richiesta autenticazione. Queste tre variabili sono passate come parametri attuali alla funzione `startVncViewer(host, port, password)`.

La funzione `startVncViewer(host, port, password)` ha il compito di avviare una nuova sessione di lavoro senza la quale sarebbe impossibile effettuare un collegamento con il Server e, quindi, stabilire la connessione remota. Come prima cosa viene creato un oggetto *WRCClientGatewayAsync*, definito dall'interfaccia che porta il medesimo nome, presente nel modulo *WRCClientGatewayAsync.java*. Questo oggetto consiste in un servizio remoto, definito con gli strumenti di *GWT*, che permette la comunicazione asincrona tra Client e Server. Tramite esso viene lanciata la funzione `startVNCClient(host, port, password)` a cui vengono passati i parametri ricevuti dalla chiamata precedente e che contengono le informazioni necessarie per stabilire la connessione con il Server oltre al parametro *startViewerResponse*, un particolare oggetto che gestisce le informazioni ricevute dalla comunicazione con il Server visualizzando graficamente i dati sul pannello *ImagePanel*.

Il corpo della funzione `startVNCClient(host, port, password)` è im-

plementato nel modulo *WRCClientGatewayImpl.java* che fa parte del package relativo al lato Server. Avviene così una prima, iniziale comunicazione, tra Client e Server. Crea innanzitutto un nuovo oggetto di tipo *HttpSession* a cui viene assegnata la sessione remota di lavoro attualmente in esecuzione. Sarà in questa sessione che il Client effettuerà la comunicazione con il Server e l'utente potrà operare in remoto sulla Macchina Virtuale. Utilizzando questo oggetto di sessione, viene quindi avviata la Macchina Virtuale a cui ci si dovrà collegare.

La funzione `openBochs(session)`, di seguito riportata interamente, ha come scopo primario quello di creare ed avviare una nuova istanza della Macchina Virtuale. Questo processo è specifico della sessione remota di lavoro attualmente attiva e la sua durata di vita coinciderà con quella della sessione. Per poter avviare la Macchina Virtuale è necessario predisporre i file di configurazione e di immagine, come ampiamente illustrato nella sezione 3.1. Viene quindi effettuata una copia sia del file immagine *win101.img*, contenente il sistema operativo da eseguire, sia del file di configurazione *win101.txt*, necessario per la Macchina Virtuale *Bochs*. Tali file verranno nominati con lo stesso identificativo della sessione remota di lavoro attiva in quel momento. Una volta che sono stati predisposti questi file, allora sarà possibile avviare la Macchina Virtuale che si predisporrà in attesa di richieste di connessione da parte del Client. In caso di errori nella copia e nella modifica dei file, o nell'esecuzione della Macchina Virtuale, verranno lanciate opportune eccezioni per informare l'utente.

Non appena la Macchina Virtuale *Bochs* è stata avviata correttamente e quindi in attesa di richieste di connessione, il corpo della funzione `startVNCClient(host, port, password)` prosegue con la creazione di un nuovo oggetto, *vncviewer*, tramite il quale sarà possibile eseguire la funzione `startNewVNCClient(host, port, password)` che, in seguito ad una catena di chiamate, lancerà direttamente i metodi presenti nel modulo *VncViewer.java* contenuto nel package che implementa e gestisce il codice del lato Server. Un ruolo di rilevante e primaria importanza è quello assunto dalla funzione `init()`. Questa funzione inizia con una lettura dei parametri necessari per poter stabilire la connessione. Assegna quindi in alcune specifiche variabili i valori dell'indirizzo (*host*) e della porta (*port*) del Server e legge la password necessaria per un'eventuale autenticazione. A seguito della lettura dei questi parametri, la funzione `connectAndAuthenticate()` decripta, se presente, la password ricevuta prima di tentare di eseguire la vera e propria autenticazione.

```
public void openBochs(HttpSession session){
  String nameVM = "win101";
  String pathVM = "C:/Users/Win101/Windows 1.01/";

  // Copia del file della Macchina Virtuale .img
  File sourceVM = new File(pathVM + nameVM + ".img");
  File targetVM = new File(pathVM + RC_SESSION_KEY + ".img");
  try {
    FileUtils.copyFile(sourceVM, targetVM);
  } catch (IOException e) {
    e.printStackTrace();
  }

  // Copia e modifica del file di configurazione .txt
  File sourceConf = new File(pathVM + nameVM + ".txt");
  File targetConf = new File(pathVM + RC_SESSION_KEY + ".txt");
  try {
    List<String> sourceConfLines = FileUtils.readLines(sourceConf);
    String newLine = "path=\"\$BXSHARE\\Windows 1.01\\\"
                    + RC_SESSION_KEY + \".img\",";
    sourceConfLines.set(9, newLine);
    FileUtils.writeLines(targetConf, sourceConfLines, true);
  } catch (IOException e) {
    e.printStackTrace();
  }

  // Esecuzione della Macchina Virtuale
  try {
    String command = "explorer \"C:/Users/Win101/Windows 1.01.bat\"";
    Process bochs = Runtime.getRuntime().exec(command);
    RC_SESSION_KEY_BOCHS = RC_SESSION_KEY;
    session.setAttribute(RC_SESSION_KEY_BOCHS, bochs);
    try {
      Thread.sleep(5000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

Codice 3.1: Funzione openBochs()

L'autenticazione è una delle prime operazioni eseguite nella fase di Handshaking. Essa è implementata da dalla funzione `tryAuthenticate()`, la prima funzione a fare un uso vero e proprio del Protocollo di comunicazione *RFB*. Al suo interno infatti viene letto il numero di versione del Protocollo ed inviato al Client. Se entrambe le versioni del Protocollo (quella da parte del Client e quella da parte del Server) saranno compatibili, si procederà con la lettura del successivo pacchetto che consiste nello schema di autenticazione utilizzato per stabilire la connessione: se non è richiesta autenticazione, la funzione termina correttamente; se invece è necessario effettuare l'autenticazione, il corpo della funzione leggerà il parametro relativo alla password e, dopo un'opportuna decriptazione, provvederà a verificare la sua correttezza. In questo caso, la funzione terminerà con il risultato dell'operazione di autenticazione che potrà essere positivo o negativo, visualizzando in questa eventualità un avviso con lo scopo di informare l'utente del fallimentare tentativo di autenticazione.

L'operazione di autenticazione termina la fase di Handshaking, alla quale fa seguito la fase di Inizializzazione in cui Client e Server si scambieranno messaggi per definire le specifiche di inizializzazione necessarie al corretto funzionamento del collegamento remoto. Questa fase è implementata dalla funzione `doProtocolInitialisation()` che non fa altro che eseguire delle chiamate ad altre specifiche funzioni implementate nel modulo del Protocollo: *RfbProto.java*.

Le funzioni `writeClientInit()` e `readServerInit()` hanno proprio lo scopo di inizializzare il Client ed il Server. In particolar modo, l'inizializzazione del Client definisce l'opzione di condivisione del Desktop; l'inizializzazione del Server definisce tutti i parametri riguardanti la dimensione del *Framebuffer* ed il formato dei pixel.
Segue poi la funzione `setEncodings()` che imposta ed invia al Server *RFB* tutte le specifiche di codifica dei dati scambiati durante la comunicazione.

Con l'impostazione delle codifiche di pixel, termina anche la fase di Inizializzazione del Protocollo *RFB*. Da questo momento in poi, se tutto si è svolto in modo corretto senza errori, la connessione è finalmente stabilita e Client e Server possono comunicare tra loro. Questa fase di Scambio dei messaggi, è implementata dalla funzione `run()` nel modulo *VncViewer.java*, la quale esegue una chiamata alla funzione `processNormalProtocol()` contenuta nel modulo *VncCanvas.java* preoccupandosi inoltre di gestire tutte le eccezioni che la normale esecuzione del Protocollo utilizzato nello scambio di messaggi

potrebbe riscontrare.

La funzione `processNormalProtocol()`, dopo un iniziale aggiornamento del Framebuffer, si mette in attesa dei messaggi ricevuti dal Server e ne esegue il processamento. Questo stato di attesa consiste in un loop `while(true)`, nel cui corpo la funzione `readServerMessageType()` riceve dal Server un valore numerico, assegnato ad una variabile di tipo *Integer*, che rappresenta il tipo di messaggio da processare. La gestione di tutti i messaggi ricevuti varia a seconda del tipo del messaggio. Il comando condizionale `switch() - case` elenca tutti gli eventuali tipi di messaggio che è possibile ricevere dal Server, che possono essere, ad esempio, richieste di aggiornamento del *Framebuffer* (`FramebufferUpdate`), impostazioni delle mappature del colore (`SetColourMapEntries`), trasferimento di file (`rfbFileTransfer`), etc.

Durante la fase di Scambio dei messaggi, l'utente opera in remoto sulla Macchina Virtuale come se lavorasse concretamente sulla postazione remota. Ogni modifica apportata al pannello della finestra in cui sono visualizzati i dati ricevuti dal Server viene eseguita sul Server stesso.

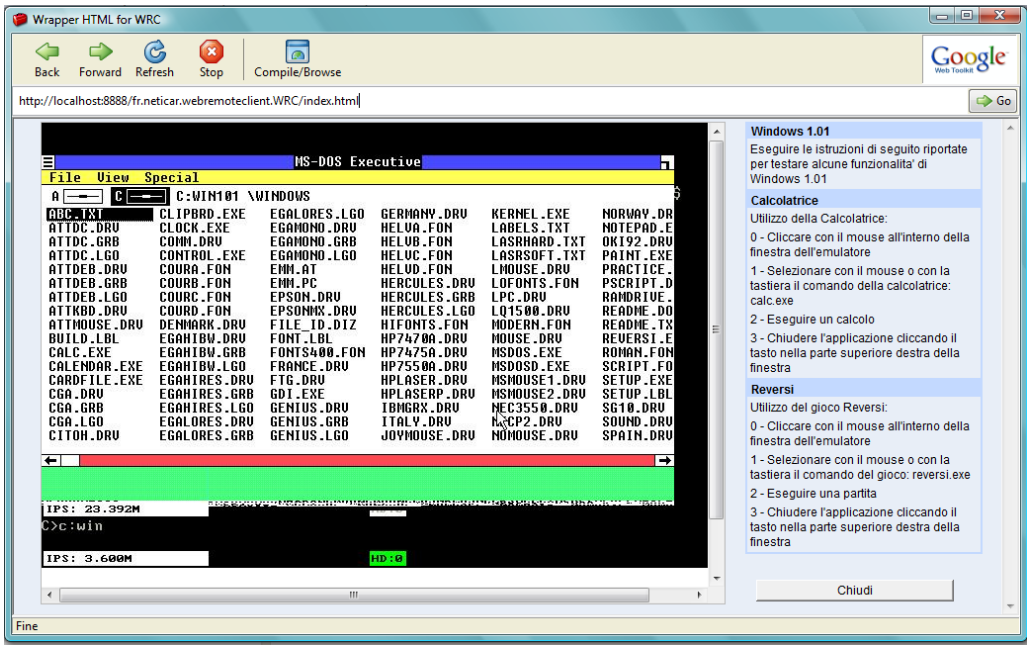

Figura 3.4: Connessione stabilita con il Server VNC. La Macchia Virtuale Bochs è visualizzata graficamente nella pagina web

Quando l'utente, dopo aver operato in modo remoto sul proprio computer, desidera porre termine al collegamento, potrà farlo grazie alla presenza di uno speciale bottone presente all'interno della pagina web del Client. L'evento della pressione sul bottone contrassegnato dall'etichetta "Chiudi", viene intercettato e gestito dal modulo *LoginPanel.java* che esegue una chiamata alla funzione `stopVncViewer()` del modulo principale *WRC.java*. Questa funzione recupera tutti i dati della sessione remota di lavoro attiva e, tramite essa, esegue la funzione `stopVNCClient(stopViewerResponse)` implementata nel modulo *WRCClientGatewayImpl.java*.

I compiti principali di questa funzione sono prevalentemente due: chiudere la sessione remota di lavoro attiva ed eliminare l'istanza della Macchina Virtuale relativa a quella stessa sessione.

La chiusura della sessione remota viene effettuata invocando al funzione `stop()` implementata nel modulo *VNCClient.java* che a sua volta chiama la funzione `disconnect()` presente nel modulo *VNCViewer.java*. Tale funzione esegue il metodo `close()` implementato nel modulo *RfbProto.java* che si occupa di chiudere i socket aperti durante la connessione ed interrompere quindi il collegamento tra Client e Server.

L'eliminazione dell'istanza della Macchina Virtuale relativa alla sessione remota di lavoro che è appena stata chiusa avviene con la chiamata alla funzione `closeBochs()`. Questa funzione elimina il processo della Macchina Virtuale specifico per l'utente, creato ed avviato nella fase iniziale del collegamento, e cancella tutti i file relativi alla sessione remota di lavoro creati ad hoc.

Una volta che l'esecuzione della funzione invocata al momento della pressione del bottone di chiusura della connessione ha avuto esito positivo, ogni file di configurazione della Macchina Virtuale viene cancellato dal Server. Viene eliminato anche il processo relativo alla simulazione della Macchina Virtuale operativa per l'utente che ha richiesto l'accesso. La terminazione di questo processo è seguita dalla chiamata alle funzioni che si occuperanno di gestire e chiudere la finestra della pagina web attiva per mezzo della quale si è potuto effettuare la connessione remota al Server.

```
public void closeBochs(){
        // Elimina il processo bochs
        HttpSession session = getSession();
        session.removeAttribute(RC_SESSION_KEY_BOCHS);

        // Cancella i file di configurazione ed immagine
        String path = "C:/Users/Win101/Windows 1.01/";
        File bochsImg = new File(path + RC_SESSION_KEY + ".img");
        File bochsConf = new File(path + RC_SESSION_KEY + ".txt");
        bochsImg.delete();
        bochsConf.delete();
}
```

Codice 3.2: Funzione closeBochs()

Conclusioni

Come ampiamente descritto nel testo, questo lavoro di tesi ha rivolto la propria attenzione sui Client *VNC* e le connessioni remote e grafiche tra due dispositivi elettronici in un contesto orientato al web, grazie all'utilizzo di strumenti di sviluppo come *GWT* e *HTML5*.

L'elaborazione del lavoro si è composta di due diverse fasi che hanno fornito una panoramica dettagliata dell'oggetto in esame. La prima fase riguarda lo studio del panorama attuale in cui si inserisce e dove si riscontra la necessità di effettuare connessioni remote sicure e stabili. Tale studio ha approfondito tutti gli strumenti e le tecnologie utilizzate arrivando a delineare in modo completo ed esaustivo gli elementi che compongono il progetto in esame. La seconda fase ha come oggetto principale la sperimentazione di un particolare software che permette la creazione e l'utilizzo del Client *VNC* nell'ambito presentato e descritto nella prima parte dell'opera.

Il primo capitolo di questa tesi ha concentrato lo studio sulle *Macchine Virtuali*, importanti software che permettono di virtualizzare un ambiente di lavoro, un software ed anche un vero e proprio computer, realizzando una perfetta emulazione delle sue componenti hardware. È stata spiegata, grazie alla presentazione di numerosi vantaggi, l'importanza delle *Macchine Virtuali* in ogni ambito dell'Informatica, da quello lavorativo a quello puramente orientato allo sviluppo ed alla ricerca. Han fatto poi da seguito la presentazione di alcune delle principali *Macchine Virtuali* sviluppate ed impiegate nell'ambito informatico, quali *VMWare*, *JVM* e *Bochs*, e l'illustrazione delle diverse modalità con cui è possibile accedere e creare un collegamento tra *Macchina Virtuale* e computer. Queste modalità d'accesso prevedono l'utilizzo di determinate regole e protocolli. Nello specifico sono stati presi in oggetto il Protocollo *RFB* ed il Protocollo *RDP*.

È proprio lo studio del Protocollo *RFB* che è stato approfondito nel secondo capitolo di questo elaborato. Il capitolo ha inizio con la presentazione di tutte le diverse fasi che compongono il Protocollo in esame, fasi che riguardo

in modo particolare la visualizzazione e la renderizzazione delle immagini a partire dai dai trasmessi durante il collegamento, le modalità con cui vengono codificati le informazioni riguardanti gli eventi di I/O e la formattazione dei pixel che rappresentano graficamente il desktop remoto con cui è stato effettuato il collegamento. La parte finale del capitolo ha riguardato l'elenco completo dei messaggi che Client e Server si scambiano per poter stabilire e terminare la connessione nonché per comunicare tra loro una volta che il collegamento è stato stabilito. Particolare attenzione è stata riservata alla fase di Handshaking, alla fase di Inizializzazione ed alla fase di Scambio dei messaggi del Protocollo *RFB*.

Sono poi stati presentati, nel terzo capitolo dell'opera, gli strumenti impiegati nella sperimentazione del Client *VNC* basato su *GWT* ed *HTML5*. La Macchina Virtuale *Bochs*, che emula il sistema operativo *Windows 1.01*, rappresenta in questo caso specifico di studio il Server *VNC* a cui il Client richiede la connessione. Tale Client *VNC* è stato sviluppato per mezzo degli strumenti offerti dalla tecnologia *GWT*, grazie alla quale è possibile realizzare un'applicazione perfettamente integrata in una pagina web definita ed implementata con lo standard *HTML5*. *WebVNC* è il progetto originario che, sviluppando il Software *VNC*, permette la connessione remota. Basandosi su tale progetto, sono state apportate alcune modifiche, illustrate in questo stesso capitolo nella parte relativa alla sperimentazione, che permettono all'utente di effettuare un collegamento remoto alla Macchina Virtuale *Bochs* che simula il sistema operativo *Windows 1.01*.

Il Client *VNC* e le modifiche apportate sono stati realizzati con il linguaggio *Java*, compilato in *JavaScript* e quindi perfettamente compatibile con ogni tipo di Browser web, grazie agli strumenti *GWT* impiegati.

Il lavoro presentato in questa tesi fa parte di un progetto più ampio. Grazie alle tecnologie approfonditamente studiate e sperimentate in questo elaborato, sarà possibile tramite una specifica pagina web del Dipartimento di Matematica e Informatica dell'Ateneo dell'Università di Parma, effettuare l'accesso alle macchine che compongono il Museo dell'Informatica situato nel Dipartimento di Matematica. L'accesso ai computer ed ai sistemi operativi patrimonio del Dipartimento permette di osservare, studiare ed anche operare direttamente con macchine e sistemi che hanno fatto la storia dell'Informatica.

Ultimo, ma non meno importante, è il vantaggio rappresentato dal più generale impiego di un Client *VNC* basato su *GWT* e *HTL5*. Tali tecnologie sono fortemente orientate al futuro, in un'epoca storica in cui l'Informatica

assume un ruolo primario in ogni ambito ed in ogni contesto della vita dell'uomo.

La possibilità di utilizzare in modo assolutamente compatibile e trasparente un'applicazione, senza il vincolo e l'obbligo di sottostare a determinate caratteristiche hardware o software, rappresenta uno dei fattori che potrebbero determinarne la riuscita vincente od il fallimento. L'oggetto di questa tesi si pone proprio questo fine: fornire un'applicazione che sia compatibile con ogni tipo di dispositivo e sistema operativo, che non richieda l'uso e l'installazione di alcun componente aggiuntivo e che garantisca la massima efficacia ed efficienza.

Ringraziamenti

Giunto a questo punto, non posso esimermi dal menzionare alcune persone che, direttamente o indirettamente, in tutto o in parte, hanno contribuito a rendere tale questa grande opera che rappresenta un passo importante e decisivo, tanto per la mia vita quanto per quella di coloro che mi sono accanto.
Il primo ringraziamento è doverosamente rivolto al mio Relatore, il Prof. *Federico Bergenti*. Se sono arrivato a scrivere questa pagina lo devo principalmente a lui, al suo preziosissimo aiuto ed alla sua infinita pazienza, alla sua disponibilità nel saper sempre rispondere alle domande, ai dubbi, alle perplessità che si sono presentate a me nel corso di questo lavoro.
Ringrazio i miei genitori, *Emanuela* e *Alessandro*, a cui dedico questa opera. Se posso affermare con orgoglio di essere ciò che sono, in un continuo divenire, lo devo esclusivamente a loro. Li ringrazio per l'educazione che mi hanno da sempre impartito e per gli insegnamenti che non mancano mai. Li ringrazio per la loro inesauribile pazienza e per avermi sempre spronato a riporre il massimo impegno in tutto ciò che mi accingo a fare. Li ringrazio per il semplice ma non irrilevante fatto di essere *mamma* e *papà*. Ringrazio anche mia sorella *Gloria*, poiché lei c'è sempre e garantisce quel tocco di allegria che rende tutto più speciale.
Grazie a *Monica*, per aver iniziato a camminare insieme a me, con me, fin dal primo giorno del mio percorso di istruzione universitaria e per non aver mai smesso. Grazie perché mi stai accanto e condividi con me la tua vita, rendendo migliore la mia.
Grazie ai miei *amici*, ai miei *compagni*, ai miei *colleghi*. A voi che, vicini di casa o lontani centinaia di chilometri, siete sempre riusciti a strapparmi una risata preziosa e sincera.
Infine rivolgo il mio ultimo ringraziamento all'*Università di Parma*, in particolar modo al *Dipartimento di Matematica e Informatica* ed a tutti i suoi Professori, la cui ammirevole cultura e preparazione riesce a trasmettere agli studenti una profonda passione verso questa splendida disciplina.

Bibliografia

[1] Paul Cotton, Sam Ruby, Maciej Stachowiak, Michael Smith, and Robin Berjon. W3c html working group, 1994.

[2] Alexandr de Pellegrin and Victor Freches. Webvnc - full ajax vnc client based on gwt, 2007.

[3] Ian Hickson. Web hypertext application technology working group, 2004.

[4] VMware Inc. Vmware virtualization for desktop and server, application, public and hybrid clouds, 2014.

[5] Adrian Nye. *Xlib Reference Manual*, volume 1. O'Really & Associates, Inc., 1992.

[6] Oracle. Oracle and sun microsystems, 2011.

[7] Greg Roelofs, Jean-loup Gailly, and Mark Adler. Zlib a massively spiffy yet delicately unobtrusive compression library, 1996-2013.

[8] Ruppert Volker. bochs: The open source ia-32 emulation project, 2001-2013.