a cura di Corrado Giustozzi

L'evoluzione di X Window sotto Linux: The K Desktop Environment e Gnome

L'interfaccia grafica X Window

Da questo mese, come avrete forse notato, la rubrica Client Computing... raddoppia. In seguito alle richieste di molti lettori che chiedevano la presenza più costante di uno spazio specificamente dedicato a Linux, abbiamo deciso di espandere quello che sinora era stato solo un... AngoLinux facendolo diventare una rubrica a sé stante, interamente dedicata a questo interessante ed ormai diffusissimo sistema operativo. La rubrica Client Computing proseguirà così a trattare regolarmente i suoi temi istituzionali mentre, dal prossimo mese, questo nuovo spazio assumerà la sua configurazione definitiva come rubrica autonoma, occupandosi sia di aspetti teorici che di aspetti pratici connessi all'uso del più famoso derivato freeware di Unix.
La rubrica sarà naturalmente portata avanti da Giuseppe Zanetti, che per tanti mesi ci ha fatto compagnia col suo AngoLinux. A Giuseppe ed alla sua nuova rubrica vanno sin d'ora i miei (e vostri) ringraziamenti e gli auguri di buon lavoro.

Corrado Giustozzi

di Giuseppe Zanetti

La prima cosa che viene notata avvicinandosi a Linux ed al suo sistema grafico X Window provenendo da un ambiente Microsoft è la mancanza di una interfaccia grafica coerente fra tutti i programmi. Al contrario di quanto succede in Windows, dove la maggior parte delle applicazioni sono scritte attorno alla loro veste grafica, nel mondo UNIX quasi tutti i programmi funzionano ancora mediante una interfaccia testuale o *a linea di comando*. Ciò è subito spiegabile se si tiene conto del fatto che la storia di UNIX ebbe inizio più di 20 anni or sono, quando la periferica standard di uscita era costituita da una telescrivente meccanica (ecco spiegato perché i file che rappresentano i terminali si chiamano */dev/tty*). Nei tempi che seguirono furono introdotti i primi terminali grafici; tuttavia non vi era un vero e proprio standard ed al più essi erano in grado di interpretare alcune sequenze di caratteri come semplici primitive grafiche. Un esempio molto diffuso di terminale grafico è il Tektronix 4014, la cui emulazione è presente anche nel programma **xterm** (è possibile abilitarla mediante il menu che si ottiene premendo il tasto centrale del mouse assieme a CTRL).

Ora il mondo ruota attorno alle interfacce grafiche, anche se, come ben sanno quelli di noi che hanno avuto la fortuna di avvicinarsi ai computer prima di una decina d'anni or sono, esse, per definizione stessa di interfaccia, creano un livello di astrazione dal funzionamento reale della macchina tale dal limitarne di molto l'utilizzo.

L'approccio interamente grafico al sistema infatti, pur permettendo l'utilizzo di certe tipologie di programmi, nasconde quello che è lo strumento più potente con cui l'utente può interagire col sistema operativo, ovvero l'interprete di comandi.

Senza una shell testuale, con relativo linguaggio di programmazione, non solo certe cose sono più complesse da realizzare, ad esempio eseguire operazioni in modalità batch, ma si è anche ogni volta costretti o ad avere un programma gigantesco in grado di compiere esattamente le operazioni che ci interessano, oppure a dover utilizzare in sequenza più programmi, la maggior parte delle volte con formati di dati che necessitano di qualche forma di conversione. L'interfaccia a caratteri di UNIX e la filosofia stessa di questo sistema operativo invece invogliano

alla scrittura di programmi semplici e che possono essere utilizzati assieme come mattoncini per creare velocemente procedure anche complesse. Ciò grazie essenzialmente alla presenza di un linguaggio di programmazione interno alla shell ed alla possibilità di inviare l'output di un programma come input ad un altro programma (*pipe*).

Presi tre piccoli comandi che compiono operazioni molto semplici, *sort* (ordina alfabeticamente le linee di un file), *uniq* (elimina le righe duplicate in un file ordinato), *nl* (scrive le righe di un file facendole precedere da un numero progressivo), è possibile utilizzarli in cascata per compiere assieme tutte le operazioni suddette:

```
sort nomefile | uniq | nl
```

In questo caso sono stati utilizzati comandi già presenti di serie nel sistema operativo, ma la vera potenza sta nel fatto che possiamo creare noi stessi dei comandi che compiano operazioni "elementari" e poi riutilizzarli per creare procedure più complesse. È possibile far ciò senza dover digitare ogni volta tutti i comandi necessari, creando uno shell script, ovvero un piccolo programma che può essere poi richiamato all'occorrenza, possibilmente con parametri diversi.

Inseriamo innanzitutto i comandi dentro il solito file "pippo":

```
#!/bin/sh
sort $1 | uniq | nl
```

Rendiamolo poi "eseguibile", assegnandogli gli opportuni permessi:

```
chmod 755 pippo
```

Ora è possibile provare a lanciarlo, semplicemente scrivendone il nome (eventualmente nella forma *./pippo* se la directory su cui l'abbiamo salvato non è compresa nella variabile d'ambiente $PATH) seguito dal file su cui vogliamo operare, che andrà a sostituire la variabile posizionale $1:

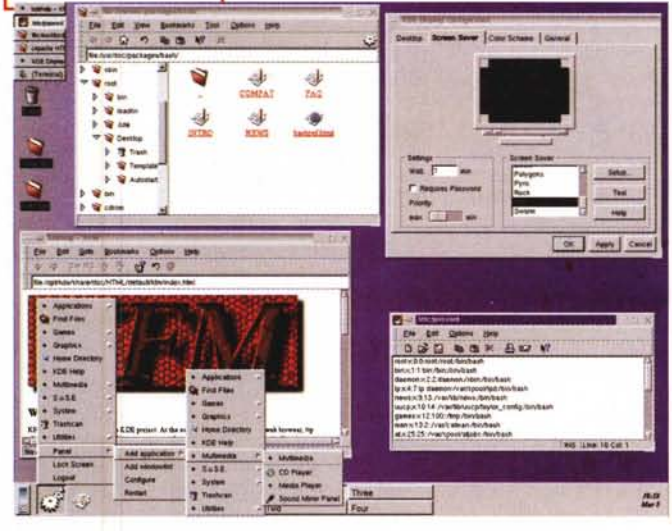
```
pippo elenco.txt
```

L'esempio appena descritto è molto banale, ma serve per rendere l'idea di come, scrivendo opportunamente gli shell script, sia possibile riutilizzarli come dei nuovi mattoncini.

Da quanto detto si capisce perché se si prova a chiedere ad un utente UNIX il motivo per cui utilizza l'interfaccia grafica, con molta probabilità egli risponderà che è un modo per avere sullo schermo più shell contemporaneamente. Se invece si pone la medesima cosa ad un utente di Windows, egli risponderà quasi sicuramente che l'interfaccia grafica è l'unica che conosce.

Kfm, il file manager di KDE convive benissimo col window manager AfterStep nel desktop più personalizzato che conosca: la scrivania di Saint.

L'ambiente KDE assieme ad alcuni dei programmi forniti a corredo.



La configurabilità dell'ambiente

Oltre a quanto detto vi è un altro fattore che ha sempre condizionato l'approccio del mondo UNIX alle interfacce grafiche: gli utenti UNIX, per cultura, si aspettano che un programma possa (e debba) essere configurato secondo i gusti e le necessità di chi lo deve utilizzare. Questo modo di pensare è l'esatto contrario di quello a cui sono abituati gli utenti di Windows, che pretendono (giustamente, dal loro punto di vista) di trovare in ogni programma una certa similarità e coerenza con l'ambiente e con gli altri programmi con cui sono soliti lavorare. Essi si aspettano inoltre che le operazioni base rimangano sempre uguali, ad esempio che per chiudere una finestra si utilizzi sempre il bottone a forma di croce oppure che nel menu File vi siano i comandi per salvare il file o terminare il programma.

Un utente UNIX, al contrario, si aspetta di poter configurare anche il minimo particolare del proprio desktop, dalla forma delle finestre al comportamento del sistema all'accadere di determinati eventi.

Alla ricerca di un'interfaccia coerente in ambiente X Windows

I fattori della configurabilità delle applicazioni e della coerenza dell'ambiente condizionano moltissimo l'accettazione di Linux da parte degli utenti Windows. Mentre il primo, come

I programmi vedono i nomi dei file come URL: in questo modo si ottiene una maggiore integrazione fra risorse locali e risorse condivise in rete. Si noti la possibilità di posizionare dove si desidera i vari menu.

abbiamo già detto, è un problema di ordine prettamente culturale, per capire il perché del secondo è necessario tornare all'inizio della storia di X Window.

Fino a pochi anni or sono per una persona che volesse programmare in ambiente X era cosa normale utilizzare una qualunque delle decine di librerie (quasi nessuna col medesimo look) presenti sul mercato o su Internet e inventarsi di sana pianta la disposizione dei menu del proprio software. Le motivazioni purtroppo sono state sempre le stesse: la mancanza di librerie standard semplici da utilizzare senza doversi studiare pagine e pagine di manuale, ed il tentativo di sopperire a questa mancanza non mediante una, bensì mediante decine di soluzioni diverse.

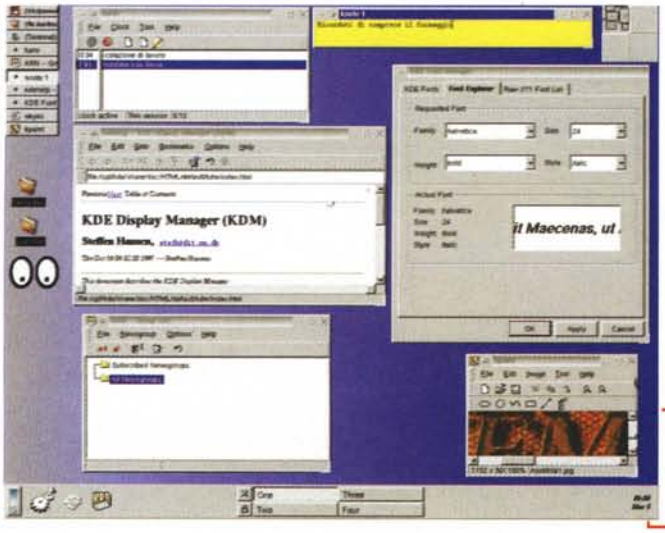
Motif

In questa rubrica stiamo parlando di Linux, e quindi di software libero, in quanto se uno ha la possibilità e la voglia di spendere dei soldi per lo sviluppo del proprio programma, può rivolgersi verso Motif, un pacchetto, formato da un insieme di librerie e *widget* (componenti per creare applicazioni grafiche, come bottoni, scrollbar, ...) e da un window manager. Motif è disponibile su tutti i sistemi UNIX, perciò anche per Linux, ed inoltre viene accettato come tool per lo sviluppo di applicazioni commerciali dalle maggiori software house.

L'utilizzo di Motif permette di scrivere programmi grafici con un aspetto simile fra loro, mentre la necessità di adeguarsi a degli standard commerciali impone al programmatore l'obbligo di seguire determinati canoni nella progettazione dell'interfaccia.

Non bastasse il fattore soldi, vi è tuttavia un altro problema che scoraggia dall'utilizzo di librerie commerciali in programmi liberi: non essendo possibile distribuire la versione linkabile dinamicamente della libreria, esistono due soluzioni. La prima, scelta ad esempio dalla suite StarOffice, consiste nel distribuire i programmi senza la libreria e lasciare all'utente il compito di procurarsela. Ma allora perché uno studente, per cui l'utilizzo del pacchetto è gratuito, dovrebbe spendere un centinaio di dollari per acquistare una libreria? E se poi magari il programma non piace e si decide di non utilizzarlo?

Il secondo approccio consiste invece nel linkare staticamente in un unico grande eseguibile la libreria al programma che si vuole distribuire. Questa è la strada seguita da Netscape. In questo caso non vi è violazione del copyright, ma è necessario tenere in memoria più copie della libreria nel caso più programmi ne facciano uso. Utilizzando una libreria caricata in modo dinamico si sarebbero invece risparmiati molti megabyte di memoria, grazie alla possibilità di caricarne una sola copia e condividerne il codice. Ultimamente iniziano ad essere reperibili librerie libere più o meno compatibili con Motif, ad esempio FreeMotif, che probabilmente saranno molto utili anche per la ricompilazione dei sorgenti di Netscape 5 ora che essi sono stati resi disponibili liberamente.



Altri programmi forniti con KDE

Le alternative free

Una conseguenza del fatto che assieme a X Window vengano resi disponibili i sorgenti di un *window manager* (twm), è che regolarmente qualcuno si sveglia alla mattina e decide di scriversi il proprio ambiente grafico. Curiosando nella directory dedicata all'argomento su *ftp://sunsite.unc.edu/pub/Linux/X11/window-managers/* possiamo trovare diversi prodotti oltre al fvwm che ormai è diventato lo standard de facto in Linux. Alcuni di essi addirittura emulano il look di altri sistemi operativi, come fvwm95, che copia il desktop di Windows 95, oppure AfterStep, che invece permette di avere una interfaccia utente simile a quella del Next.

KDE e Gnome

Le due novità del momento non si limitano ad offrire un nuovo ambiente di lavoro, ma tentano di risolvere una volta per tutte i problemi a cui abbiamo accennato, fornendo, oltre che il window manager, anche una serie di "oggetti da scrivania" e utilità coerenti fra loro e delle librerie per sviluppare nuovi programmi. Esse sono KDE, acronimo di "The K Desktop Environment" (*http://www.kde.org/*) e Gnome, che significa invece "GNU Network Object Model Environment" (*http://www.gnome.org/*).

Entrambi i prodotti sono disponibili gratuitamente, ma il primo ha il difetto di appoggiarsi su una libreria, Qt, che, seppur in grado di offrire "il meglio" attualmente disponibile sul mercato, non è libera. Essa infatti non è utilizzabile gratuitamente per creare programmi commerciali ed anche nel caso di programmi liberi è necessario comunicare preventivamente i propri dati e ciò che si vuole fare alla ditta che l'ha scritta. Inoltre, i programmi che usano Qt non possono includere codice prove-

Come funziona X Window

X Window System, familiarmente X o X11, è il sistema grafico standard per l'ambiente UNIX. Esso è stato creato nell'ambito del Progetto Athena del *MIT* (Massachusetts Institute of Technology). Alla base dell'intero sistema vi è una libreria grafica che mette a disposizione del programmatore dei metodi per scrivere software grafico in maniera indipendente dall'hardware su cui esso deve funzionare.

X è interamente basato su un'architettura di tipo client-server, in cui esistono processi in grado di fornire servizi (*server*) ad altri che li utilizzano (*client*). Essi comunicano fra loro appoggiandosi sul protocollo standard di rete TCP/IP oppure, nel caso client e server risiedano sulla stessa macchina, mediante i *socket* di UNIX.

L'aver implementato come un server lo stesso programma (*X server*) che si occupa di gestire lo schermo, la tastiera ed il mouse (questi oggetti, visti come un insieme, prendono in X la denominazione di *display*) rende possibile l'interazione fra programma e utente su una macchina diversa da quella su cui il programma effettivamente gira. L'utilizzo in modalità *display remoto* permette, ad esempio, di aprire una applicazione su un computer a Padova e di interagire con esso sullo schermo di un computer portatile in Alaska, indipendentemente dal fatto che il paesaggio attorno a noi mostri le cupole della basilica del Santo oppure desolate distese di ghiaccio.

Il fatto che sul 99% dei sistemi il display utilizzato sia quello della macchina locale non deve trarre in inganno, in quanto lo stesso identico risultato sarebbe ottenibile anche nascondendo sotto la scrivania, oltre al computer, un *X terminal*, ovvero un terminale grafico basato su un hardware a sé stante, che dialoga col PC attraverso la rete. In effetti *X server* e *X terminal* svolgono la medesima funzione, che è semplicemente quella di creare le finestre ed i disegni grafici in funzione dei comandi che la libreria invia mediante un determinato protocollo. Oltre a ciò essi si occupano di comunicare al programma, mediante opportuni *messaggi*, le azioni dell'utente, come lo spostamento del mouse o la pressione di un tasto in corrispondenza di un widget.

La procedura per aprire sul proprio display (pippo) una sessione di lavoro di un'altra macchina (pluto) è molto semplice: è sufficiente far partire X sulla propria macchina locale e scrivere il comando per abilitare l'accesso al proprio display da parte della macchina remota:

```
pippo:/# xhost + pluto
```

A questo punto si deve fare un telnet sull'altra macchina e settare correttamente il valore della variabile d'ambiente che indica il display da utilizzare:

```
pluto:/# DISPLAY=pippo:0
pluto:/# export DISPLAY
```

Infine si può eseguire un qualunque programma grafico affinché questo appaia sul display di pippo:

```
pluto:/# xterm &
```

In alternativa è possibile passare direttamente il nome del display da utilizzare sulla linea di comando dell'xterm:

```
pluto:/# xterm -display pippo:0 &
```

In entrambi i casi tutti i comandi eseguiti dalla finestra dell'*xterm* erediteranno la variabile d'ambiente DISPLAY ed appariranno sullo schermo corretto.

La libreria X e l'*X server* (o *X terminal*) non sono però sufficienti, in quanto senza un apposito programma che si occupi di implementare la *GUI* (Graphic User Interface) le finestre apparirebbero senza bordo e senza i widget necessari per chiuderle, spostarle o ridimensionarle. A questo ci pensa un apposito programma, il *window manager* (gestore di finestre), che determina, oltre allo stile con cui vengono disegnate le finestre, anche la risposta dell'ambiente in funzione delle azioni dell'operatore.

niente da programmi coperti da GPL, la licenza di GNU secondo cui è distribuito Linux stesso e la maggior parte del software per Linux, perché la violerebbero. Questo è il motivo per cui l'intero progetto non viene visto di buon occhio dalla maggior parte del mondo Linux, che invece appoggia Gnome, nonostante esso sia molto indietro come sviluppo rispetto al concorrente.

Lo scopo dichiarato di Gnome è infatti quello di "essere simile a KDE, ma basato interamente su software libero". Notate come ho sempre cura di tradurre il termine "free" in "libero", invece che nel più limitativo "gratuito".

La cosa che salta subito all'occhio visitando i siti Internet dei due progetti è la presenza di *fogli di stile*, che specificano punto per punto come scrivere un programma. La cosa interessante è che, nonostante i due ambienti siano in conflitto filosofico fra loro, la guida allo stile di Gnome invita i programmatori a scrivere oggetti che si trovino "come a casa propria" anche nella scrivania di KDE.

Gli scopi dei due progetti sono a prima vista simili e consistono nel creare una interfaccia utente allo stesso tempo bella, coerente, semplice da utilizzare e al passo con le nuove tendenze che Windows impone per il desktop (*drag & drop* fra file e oggetti della scrivania, accesso mediante un'unica interfaccia alle risorse locali ed alle funzionalità di rete, integrazione del desktop con Internet, tool di configurazione dell'ambiente amichevoli, ...). Entrambi i prodotti inoltre forniscono delle librerie ed un insieme di widget per creare i propri programmi mantenendo stile e funzionalità coerenti.

Tuttavia, mentre KDE si limita all'interfaccia, Gnome la considera invece solamente la base per un progetto più ambizioso, ovvero la creazione di un ambiente standard ad oggetti per la piattaforma UNIX, con alla base i seguenti componenti:

- CORBA (Common Object Request Broker Architecture);
- GTK (Gimp ToolKit);
- Guille.

Di queste, la tecnologia senz'altro più importante è CORBA (*http://www.omg.org/about/wicorba.htm*), una interfaccia standard, non proprietaria, che permette di invocare metodi su oggetti in modo indipendente dalla locazione degli stessi nella rete e dall'architettura della macchina su cui essi funzionano. Delle diverse implementazioni di CORBA è stata scelta MICO, principalmente per motivi di licenza d'uso.

Come libreria grafica e insieme di widget è stato scelto GTK, il toolkit su cui è stata modellata l'interfaccia del programma Gimp. Esso non è certamente ai livelli qualitativi di Qt, specialmente per quanto riguarda la documentazione, ma ha il pregio di essere libera e con molte potenzialità di crescita.

L'ultimo componente, Guille, è l'implementazione GNU di Scheme, un dialetto di Lisp, che dovrebbe essere utilizzato assieme a GTK per lo sviluppo di piccole utilità e applicazioni. A detta degli autori, Guille dovrebbe diventare per Gnome l'equivalente di Visual Basic per Windows.

Altri componenti che dovrebbero in futuro entrare a far parte del progetto sono Display Postscript e OpenDoc.

MC