X Window System Architecture Overview HOWTO

Daniel Manrique

roadmr@entropia.com.mx

Diario delle Revisioni

Revisione 1.0.1 2001-05-22 Revisionato da: dm
Alcune correzioni grammaticali, indicate da Bill Staehle
Revisione 1.0 2001-05-20 Revisionato da: dm
Release iniziale LDP.

Questo documento fornisce una panoramica dell'architettura del sistema X Window, aiuta a comprendere meglio la sua progettazione e spiega quali componenti si integrano con X e collaborano per formare un ambiente grafico. Inoltre illustra le scelte a disposizione per quanto riguarda componenti come window manager, librerie, toolkit di widget e ambienti desktop. Traduzione a cura di Silvio Donnini, e-mail: scaudi at alice dot it Revisione a cura di Giulio Daprelà e-mail: daprela at pluto dot it

1. Prefazione

Questo documento fornisce una panoramica dell'architettura del sistema X Window, aiuta a comprendere meglio la sua progettazione e spiega quali componenti si integrano con X e collaborano per formare un ambiente grafico. Inoltre illustra le scelte a disposizione per quanto riguarda componenti come window manager, librerie, toolkit di widget e ambienti desktop.

Indaghiamo su diversi concetti che vengono menzionati spesso ma potrebbero risultare un po' oscuri per coloro che non hanno un background tecnico riguardante widget, toolkit, window manager e ambienti desktop. Vengono forniti alcuni esempi di come questi componenti interagiscono durante l'uso quotidiano delle applicazioni.

Questo documento è deliberatamente non troppo tecnico. È basato sulla conoscenza empirica dell'autore sull'argomento, e, benché sia inteso principalmente come un'introduzione non tecnica, sono bene accetti tutti i tipi di commenti, esempi, spiegazioni e correzioni tecniche al riguardo. Tutte le domande e i commenti su questo documento sono benvenuti e l'autore può essere raggiunto tramite l'indirizzo *roadmr@entropia.com.mx* (mailto:roadmr@entropia.com.mx).

2. Introduzione

Ai tempi in cui Unix era cosa nuova, intorno al 1970, le interfacce grafiche erano solo stranezze con cui si giocherellava nei laboratori (lo Xerox's PARC per la precisione). Al giorno d'oggi, comunque, ogni sistema operativo che vuole essere un minimo competitivo ha bisogno di un sottosistema GUI (Infaccia Grafica per l'Utente). Le GUI sono ritenute più facili da usare. Ma questo non interessa granché ad un utente di Unix, sistema che è sempre stato, tradizionalmente e in un certo suo modo, abbastanza insensibile alle esigenze di usabilità dei propri utenti, preferendo la versatilità alla facilità d'uso. Tuttavia ci sono parecchi motivi per cui una GUI è desiderabile anche su un sistema Unix. Per esempio, data la natura multitasking di Unix, è naturale avere in ogni momento molti programmi che girano sulla stessa macchina. Una GUI dà più controllo su come i programmi sono visualizzati sullo schermo, fornendo strumenti per gestire moltissimi programmi in contemporanea. E poi alcune informazioni rendono di più in formato grafico (alcune, addirittura, possono essere visualizzate solo in forma grafica; come il porno e altri dati intrinsecamente grafici).

Storicamente Unix ha ricevuto molti miglioramenti provenienti da ambienti accademici. Un buon esempio è il codice di rete di BSD, aggiunto alla fine degli anni '70, che era, ovviamente, il risultato del lavoro svolto all'università della California, a Berkeley. Anche il sistema X Window (anche detto X, ma mai X Windows), che rappresenta le fondamenta per la maggior parte dei sottosistemi GUI dei sistemi Unix moderni (inclusi Linux e i vari BSD), è il prodotto di un progetto universitario, ovvero il progetto Athena del Massachusetts Institute of Technology (MIT).

Unix è sempre stato fin dai suoi inizi multiutente, multitasking e time sharing. Inoltre, da quando vi sono state incorporate tecnologie di rete, ha avuto la capacità di permettere a un utente di connettersi da remoto e lavorare sul sistema. Precedentemente ciò era fattibile o collegandosi via terminale seriale o attraverso una connessione di rete (il leggendario telnet).

Quando giunse il tempo di sviluppare un sistema GUI che potesse girare principalmente sotto Unix, questi concetti vennero tenuti a mente e incorporati nella progettazione. In realtà X ha una struttura piuttosto complessa, cosa che è spesso stata menzionata come uno svantaggio. Tuttavia, proprio grazie alla sua struttura, esso è anche un sistema molto versatile, e ciò diverrà molto chiaro quando spiegheremo come si incastrano fra loro le parti che vanno a comporre una GUI.

Prima di andare a vedere l'architettura di X, è necessario parlare un po' della sua storia e di come esso sia arrivato sui sistemi Linux.

X è stato sviluppato dal progetto Athena e rilasciato nel 1984. Nel 1988 un ente chiamato "Consorzio X" prese le redini del progetto, e ad oggi gestisce il suo sviluppo e la sua distribuzione. Le specifiche di X sono disponibili al pubblico, mossa saggia che ha reso X onnipresente. Ecco come venne alla luce XFree86: XFree86 è l'implementazione di X sulle nostre macchine Linux. XFree86 funziona anche su altri sistemi operativi, come i vari *BSD, OS/2 e forse altri. Inoltre, nonostante il suo nome, XFree86 è disponibile per diversi tipi di processore.

3. Architettura del sistema X Window: una panoramica

L'architettura di X è client-server. Le applicazioni stesse sono dei client; esse comunicano con il server e inviano delle richieste, ricevendo informazioni dal server.

Il server X ha il controllo esclusivo dello schermo e dei servizi richiesti dai client. A questo punto i vantaggi di questo modello sono abbastanza chiari. Le applicazioni (client) hanno solo bisogno di sapere come comunicare con il server e non si devono preoccupare dei dettagli del dispositivo grafico fisico. Al livello base, un client dice al server cose del tipo "disegna una linea che va da qui a qui", oppure "visualizza questo testo, usando questi caratteri, in questo punto dello schermo".

È come se stessimo usando una libreria grafica per scrivere la nostra applicazione. Tuttavia il modello di X fa un passo in più. Non si limita a poter essere usato solo da un client che risiede sulla stessa macchina del server. Il protocollo usato per far comunicare client e server può funzionare anche attraverso una rete, e in realtà qualsiasi "meccanismo di comunicazione inter-processo che fornisca un flusso di byte affidabile". Ovviamente il modo preferito di far comunicare un client e un server remoto è attraverso i protocolli TCP/IP. Evidentemente il modello di X è veramente potente; l'esempio classico è quello in cui si fa girare un'applicazione che impegna pesantemente il processore su un computer Cray, un'applicazione che gestisce un database su un server Solaris, un'applicazione di posta elettronica su un mail server BSD, un programma di visualizzazione su un server SGI e poi si visualizza tutto sullo schermo di una workstation Linux.

Fin qui abbiamo visto che il server X è quello che si occupa della visualizzazione vera e propria. E, siccome è il server X che gira sulla macchina fisica su cui l'utente sta lavorando, è responsabilità del server X gestire tutta l'interazione effettiva con l'utente. Incluso leggere i movimenti del mouse e l'input della tastiera. Tutte queste informazioni sono passate al client, che ovviamente dovrà reagire ad esse.

X fornisce una libreria, chiamata Xlib, che gestisce tutte le comunicazioni client-server di basso livello. Sembra ovvio quindi che il client debba invocare le funzioni contenute in Xlib per fare quello che deve fare.

A questo punto tutto sembra andare per il verso giusto. Abbiamo un server che si occupa dell'output visivo e dell'input, applicazioni client e un meccanismo per farle comunicare tra loro. Nel figurarsi un'interazione ipotetica tra un client e un server, il client potrebbe chiedere al server di farsi assegnare un'area rettangolare dello schermo. Essendo un client, non mi interessa dove vengo messo sullo schermo. Dico solo al server: "dammi un area di dimensioni X per Y in pixel" e poi chiamo funzioni per eseguire azioni del tipo "disegna una linea da qui a qui", "dimmi se l'utente sta muovendo il mouse sopra la mia area dello schermo", e così via.

4. I Window Manager

Ad ogni modo non abbiamo mai menzionato come faccia il server X a gestire la manipolazione delle aree di visualizzazione dei client su schermo (chiamate anche finestre). È ovvio, a chiunque abbia usato una GUI, che bisogna avere il controllo su delle "finestre client". Tipicamente l'utente può muoverle e ordinarle, cambiarne le dimensioni, minimizzarle o massimizzarle. Come fa a gestire tali compiti il server X? La risposta è: non lo fa.

Uno dei principi fondamentali di X è: "noi forniamo il meccanismo, ma non la politica". E così, benché il server X fornisca un modo (meccanismo) per manipolare le finestre, non dice come si deve comportare effettivamente questa manipolazione (politica).

Tutta questa roba strana riguardo a meccanismi e politiche si riduce ad un solo precetto: è responsabilità di un altro programma gestire lo spazio su schermo. Questo programma decide dove piazzare le finestre, fornisce il meccanismo per far specificare all'utente l'aspetto delle finestre, le posizioni e la loro dimensione; solitamente fornisce "decorazioni" come i titoli delle finestre, cornici e pulsanti che ci danno il controllo della finestra stessa. Questo programma, che gestisce le finestre, è chiamato "window manager".

"Il window manager in X è solo un altro client -- non è parte del sistema X Window, benché goda di privilegi speciali -- qundi non esiste un unico window manager; ce ne sono molti, che supportano diverse modalità di interazione con l'utente e diversi schemi di posizionamento delle finestre, decorazioni, gestione della tastiera, della mappa colori e del focus."

L'architettura X fornisce i mezzi ad un window manager per eseguire tutte queste azioni sulle finestre; ma non fornisce un vero e proprio window manager.

Ci sono, ovviamente, moltissimi di window manager, poiché, dal momento che il window manager è un componente esterno, è (relativamente) facile scriverne uno che si adatti alle proprie esigenze, ovvero le proprie preferenze sul look delle finestre, il loro comportamento, la posizione in cui si vuole che si trovino, etc. Alcuni window manager sono semplicistici e bruttini (twn); alcuni hanno un aspetto vivace e includono praticamente tutto a parte un lavandino (enlightenment); poi ci sono tutte le vie di mezzo: fvwm, amiwm, icewm, windowmaker, afterstep, sawfish, kwm e tantissimi altri. Ci sono window manager per tutti i gusti.

Un window manager è un "meta-client", il cui obbiettivo di base è quello di gestire altri client. La maggior parte dei window manager fornisce qualche strumento aggiuntivo (e alcuni ne forniscono tantissimi). Comunque una funzionalità che sembra presente nella maggior parte dei window manager è un sistema per lanciare le applicazioni. Alcuni forniscono un box dove si possono scrivere comandi standard (che possono essere usati per lanciare applicazioni client). Altri hanno dei rifiniti menu appositi di qualche tipo. Non ci sono standard in proposito comunque; di nuovo, poiché X non detta alcuna politica su come una applicazione client dovrebbe essere lanciata, questa funzionalità va implementata nei programmi client. Mentre tipicamente di ciò si occupa il window manager (e ciascuno l fa in modo diverso), è concepibile avere applicazioni client il cui solo scopo è lanciare altre applicazioni client, si pensi ad un pannello di lancio dei programmi. E di certo sono state scritte molte applicazioni per lanciare programmi.

5. Applicazioni client

Spostiamo per un momento l'attenzione sui programmi client. Si pensi di voler scrivere un programma client da zero, usando solo gli strumenti forniti da X. Ci si accorgerebbe presto che Xlib è un tool abbastanza spartano, e che fare cose come posizionare pulsanti sullo schermo, testo e controlli elaborati (barre si scorrimento e pulsanti radio) è incredibilmente complicato.

Fortunatamente, qualcun altro si è preso la briga di implementare questi controlli e di presentarceli in forma utilizzabile: una libreria. Questi controlli sono conosciuti come "widget" e ovviamente la libreria è chiamata "widget library". Quindi io devo solo richiamare una funzione di questa libreria con qualche parametro e ottenere che un pulsante sia visualizzato sullo schermo. Esempi di widget includono menu, pulsanti, pulsanti radio, barre di scorrimento e canvas.

Un "canvas" (tela per dipinti) è un tipo di widget interessante, dal momento che è essenzialmente una sottoarea dell'applicazione client in cui si può disegnare. Comprensibilmente, poiché non si dovrebbe usare Xlib direttamente (perché ciò interferirebbe con il lavoro della widget library) la libreria stessa fornisce un modo di fare dei disegni arbitrari all'interno del widget canvas.

Poiché è la widget library che si occupa di disegnare realmente gli elementi sullo schermo, così come di interpretare le azioni dell'utente e trasformarle in input per le applicazioni, la libreria usata è largamente responsabile per l'aspetto e il comportamento di ogni client. Dal punto di vista di uno sviluppatore, una libreria widget ha anche le sue API (insiemi di funzioni), ed è questo aspetto che potrebbe farlo scegliere tra una particolare widget library e un'altra.

6. Widget library o toolkit

La widget library originale, sviluppata per il progetto Athena, si chiama prevedibilmente Athena widget library, conosciuta anche come Athena widgets. È molto primitiva, molto brutta e non è intuitiva per gli standard odierni (per esempio, una barra di scorrimento o uno slider non si possono trascinare: per scorrere in su o in giù si devono cliccare rispettivamente il pulsante destro e quello sinistro). Come tale non viene granché utilizzata ai giorni nostri.

Proprio come accade per i window manager, ci sono molti toolkit, progettati per scopi differenti. Uno dei primi toolkit è il famoso Motif, che faceva parte dell'ambiente grafico Motif della Open Software Foundation, e consiste in un window manager e di un toolkit. In questo documento non tratteremo la storia della OSF. Il toolkit Motif, essendo superiore ad Athena widgets, divenne di largo utilizzo tra gli anni '80 e i primi '90.

Ai giorni nostri Motif non è una scelta popolare. Non è software libero, e ci vuole denaro per ottenere una licenza per sviluppare con OSF Motif (ovvero compilare i propri programmi con esso), benché si possa distribuire liberamente un file binario linkato dinamicamente a Motif. Forse l'applicazione Motif più conosciuta, ameno per gli utenti Linux, è Netscape Navigator/Communicator (precedente a Mozilla).

Per un po' Motif è stato il solo toolkit decente disponibile, e c'è molto software Motif in giro. Ovviamente si è iniziato a sviluppare delle alternative, e oggi ci sono molti toolkit, come XForms, FLTK, e altri.

Non si sente molto parlare di Motif al momento, in particolare nel mondo del software libero. La ragione è che ora ci sono alternative migliori, in termini di licenza, prestazioni (Motif è ritenuto molto esoso in termini di risorse) e funzioni offerte.

Uno di questi toolkit, il famoso e usatissimo Gtk, è stato creato specificamente per rimpiazzare Motif nell'ambito del progetto GIMP (una possibile interpretazione di "Gtk" è "GIMP Toolkit", sebbene, data la sua enorme diffusione, potrebbe anche significare "GNU Toolkit"). Gtk adesso è molto popolare, perché relativamente leggero, ricco di funzioni, estendibile, ed è per intero software libero. La release 0.6 di GIMP includeva nel changelog l'affermazione: "Bloatif has been zorched" ("Bloatif" è un gioco di parole fra bloat (pesante) e Motif, "zorched" deriva da "zorch", che nel dialetto hacker ha un significato che varia molto a seconda del contesto. Qui suona come "il pesante Motif è stato eclissato" n.d.T.). Questa frase è il testamento della pesantezza di Motif.

Un altro toolkit molto popolare oggi è Qt. Non era molto conosciuto prima dell'avvento del progetto KDE, che utilizza Qt per tutti gli elementi della sua GUI. Di certo non affronteremo la questione della licenza d'uso di Qt, né la dicotomia KDE/GNOME. Abbiamo parlato di più di Gtk perché la sua storia come rimpiazzo di Motif è interessante; Qt viene menzionato brevemente perché è estremamente popolare.

Infine, un'altra alternativa degna di nota è LessTif. Il nome è un gioco di parole con Motif (La "Mo" di Motif viene pronunciata con un suono che somiglia a "more" (più), quindi la parola LessTif è creata sostituendo "Mo" con "Less" (meno) n.d.T.), LessTif mira ad essere un'alternativa (libera e compatibile dal punto di vista dell'interfaccia di programmazione) a Motif. Non è chiaro fino a che punto LessTif miri ad essere usato per progetti a sé, piuttosto che per aiutare coloro che hanno già del codice scritto utilizzando Motif a passare immediatamente ad un'alternativa libera mentre pianificano il passaggio ad un altro toolkit.

7. Cosa abbiamo finora

A questo punto abbiamo un'idea dell'architettura client-server di X, dove i client sono i nostri programmi applicativi. Con questo sistema grafico client-server possiamo usare parecchi window manager, che gestiscono tutto quello che viene visualizzato sullo schermo; abbiamo anche le nostre applicazioni client, che sono ciò che comunemente usiamo per lavorare. Tali applicazioni possono essere state programmate utilizzando molti toolkit diversi.

Qui le cose cominciano a farsi complicate. Ogni window manager ha un approccio diverso alla gestione dei client; il comportamento e le decorazioni variano. Inoltre, anche i client possono comportarsi ed essere visualizzati in maniera diversa tra loro a seconda del toolkit usato. Poiché non c'è niente che dica che gli sviluppatori debbano usare lo stesso toolkit per ogni loro applicazione, è perfettamente plausibile che un utente stia lavorando con diciamo sei diverse applicazioni, ognuna scritta con un toolkit diverso, e che tutte si comportino e vengano visualizzate diversamente. Ciò crea confusione perché il comportamento delle applicazioni non è consistente. Se si è mai usato un programma scritto con Athena widgets si noterà che non è molto simile a qualunque altra cosa scritta con Gtk. E si ricorderà la difficoltà di usare tutte queste applicazioni con un look & feel diverso. Questi difetti annullano il vantaggio di avere a disposizione un'interfaccia grafica.

Da un punto di vista più tecnico, usare molti toolkit diversi aumenta l'utilizzo delle risorse. I sistemi operativi moderni supportano il concetto di librerie condivise dinamicamente (dynamic shared libraries). Ciò significa che se ho due o tre applicazioni che usano Gtk come libreria linkata dinamicamente, quelle due o tre applicazioni condivideranno la stessa copia di Gtk, sia sul disco che in memoria: non c'è spreco di risorse. Se invece ho un'applicazione Gtk, un'applicazione Qt, qualcosa che usa Athena, un programma basato su Motif come Netscape, un programma che usa FLTK e un altro che usa XForms, sto caricando sei diverse librerie in memoria, una per ciascun toolkit. Bisogna tenere presente che tutti i toolkit forniscono in definitiva le stesse funzionalità.

Ci sono altri problemi. Il modo in cui si lanciano i programmi varia da un window manager all'altro. Alcuni hanno un gradevole menu per lanciare le applicazioni, altri no e si aspettano che l'utente apra un box per lanciare comandi, o usi una determinata combinazione di tasti, o ancora apra un terminale xterm e lanci tutte le applicazioni richiamandole da riga di comando. Ancora una volta non ci sono standard e le cose si fanno complicate.

Infine ci sono alcune caratteristiche utili che ci aspettiamo da una GUI e che fin qui non abbiamo trattato. Cose come un'utility di configurazione (o "pannello di controllo") o un file manager grafico, che certamente possono essere scritte come applicazioni client. E in tipico stile free software ci sono centinaia di file manager e centinaia di programmi per la configurazione del sistema, che prevedibilmente apportano ulteriore confusione al già difficile compito di avere a che fare con molti componenti software diversi.

8. Per fortuna esistono gli ambienti desktop

Ecco dove si inserisce il concetto di ambiente desktop. L'idea è che un ambiente desktop fornisce un insieme di strumenti e linee guida per standardizzare tutto ciò che abbiamo menzionato in modo da minimizzare i problemi citati.

Il concetto di ambiente desktop è nuovo per chi si avvicina a Linux per la prima volta, poiché altri sistemi operativi (come Windows e Mac OS) lo danno per scontato. Per esempio MacOS, che è una delle interfacce grafiche più antiche, fornisce un look & feel molto consistente attraverso tutta la sessione di utilizzo del sistema. Il sistema operativo fornisce molte delle caratteristiche utili menzionate: un file manager predefinito (il finder), un pannello di controllo che gestisce l'intero sistema, un singolo toolkit che deve essere usato da tutte le applicazioni (in modo che tutte vengano visualizzate allo stesso modo). Le finestre delle applicazioni sono gestite dal sistema (per essere precisi dal window manager). E infine ci sono delle linee guida che dicono agli sviluppatori come si dovrebbero comportare le proprie applicazioni, raccomandano disposizione e visualizzazione dei componenti, e suggeriscono comportamenti consistenti con le altre applicazioni del sistema. Tutto ciò per ottenere una maggiore uniformità e facilità d'uso.

A questo punto sorge una domanda: perché gli sviluppatori di X lo hanno progettato in maniera così particolare? Ha senso chiederselo, perchè si sarebbero evitati tutti i problemi menzionati in precedenza. La risposta è che nel progettare X i suoi creatori hanno scelto di renderlo il più possibile flessibile. Tornando al paradigma politica/meccanismo, MacOS fornisce principalmente politiche. I meccanismi ci sono, solo che gli sviluppatori non vengono incoraggiati ad occuparsene. Il risultato è che si perde in versatilità: se non mi piace il modo in cui MacOS gestisce le finestre o il toolkit non ha le funzioni di cui ho bisogno, non posso farci niente. Ciò non accade sotto X, benché come visto il prezzo della flessibilità sia una maggiore complessità.

Sotto Linux/Unix e X si riduce tutto a mettersi d'accordo su qualcosa e continuare ad usarlo. Prendiamo ad esempio KDE. KDE include un solo window manager (kwm), che gestisce e controlla il comportamento delle nostre finestre. Raccomanda l'utilizzo di un certo toolkit grafico (Qt), in modo che ogni applicazione KDE venga visualizzata allo stesso modo fintanto che essa resta sullo schermo. KDE estende ulteriormente Qt fornendo un insieme di librerie specifiche per l'ambiente (kdelibs) che servono per eseguire operazioni frequenti come creare menu, finestre "about", barre degli strumenti, comunicare tra programmi diversi, stampare, selezionare file ecc. Queste rendono il lavoro del programmatore più facile e standardizzano il modo in cui si comportano queste funzioni speciali. KDE fornisce inoltre un insieme di linee guida per la progettazione ai programmatori, con l'intenzione di rendere uniforme il comportamento e gli aspetti visivi delle applicazioni di coloro che le seguono. Infine, KDE fornisce, come parte dell'ambiente, un pannello di lancio (kpanel), un file manager standard (che è al momento Konqueror) e un'utility di configurazione (pannello di controllo) da cui si possono controllare molti aspetti del proprio sistema, da impostazioni come lo sfondo del desktop e il colore della barra del titolo delle finestre alla configurazione dell'hardware.

Il pannello KDE è l'equivalente della barra delle applicazioni di MS Windows. Fornisce un punto d'accesso centrale da cui lanciare applicazioni, e permette anche a piccole applicazioni, chiamate "applet", di venire visualizzate al suo interno. Ciò include funzionalità come il piccolo orologio senza il quale la maggior parte degli utenti non può vivere.

9. Ambienti desktop specifici

Abbiamo usato KDE come esemio, ma non è assolutamente il primo ambiente desktop per i sistemi Unix. Forse uno dei primi è CDE (Common Desktop Environment), un altro parente di OSF. A quanto dice la FAQ di CDE: "Common Destktop Environment è un desktop standard per Unix, che fornisce i suoi servizi all'utente finale, all'amministratore di sistema e agli sviluppatori di applicazioni su molte piattaforme." La chiave qui è la consistenza. Tuttavia CDE non era così facile e ricco di caratteristiche come invece avrebbe dovuto. Insieme a Motif, CDE è praticamento scomparso dal mondo del software libero, essendo stato rimpiazzato da alternative migliori.

Sotto Linux gli ambienti desktop più popolari sono KDE e GNOME, ma non sono i soli. Una veloce ricerca su Internet rivelerà una mezza dozzina di ambienti desktop: GNUStep, ROX, GTK+XFce, UDE, per nominarne alcuni. Tutti forniscono le funzionalità di base che abbiamo menzionato in precedenza. GNOME e KDE hanno avuto il supporto maggiore sia dalla comunità che dall'industria, cosicché essi sono i più avanzati e mettono a disposizione dell'utente e delle applicazioni una grande quantità di servizi.

Abbiamo menzionato KDE e i componenti che forniscono servizi specifici sotto tale ambiente. Da buon ambiente desktop, GNOME è simile sotto questo punto di vista. La differenza più ovvia è che GNOME non detta l'utilizzo di alcun window manager (nel modo in cui KDE fa con kwm). Il progetto GNOME ha sempre cercato di rimanere agnostico rispetto alla questione dei window manager, riconoscendo che molti utenti rimangono molto affezionati al proprio window manager preferito, e che forzarli ad usare qualcosa che gestisce le finestre in modo diverso li allontanerebbe. In principio GNOME favoreggiava il window manager Enlightenment e al momento il loro window manager preferito è Sawfish, ma il pannello di controllo di GNOME ha sempre avuto una sezione che permetteva di selezionare un window manager.

Oltre a ciò, GNOME usa il toolkit Gtk e fornisce un insieme di funzioni di alto livello attraverso l'insieme di librerie gnome-libs. GNOME ha il suo insieme di linee guida per la programmazione per consentire un comportamento consistente tra tutte le applicazioni supportate, fornisce un pannello (chiamato semplicemente "pannello"), un file manager (gmc, benché probabilmente sarà sostituito da Nautilus), e un pannello di controllo (il centro di controllo gnome).

10. Come si integra il tutto

Ogni utente è libero di usare l'ambiente desktop che preferisce. Il risultato finale è che se si usa un sistema con solo KDE o solo GNOME, allora il look & feel dell'ambiente è molto consistente e tutte le applicazioni interagiscono molto bene tra loro. Ciò semplicemente non era possibile quando le applicazioni erano scritte con una miriade di diversi toolkit. La moltitudine di strumenti forniti dai moderni ambienti desktop sotto Linux permette altri simpatici trucchetti, come architetture a componenti (KDE ha Kparts e GNOME usa il framework di componenti Bonobo), che fanno sì che si possano creare cose come documenti testuali che contengono un foglio di calcolo o un grafico, strumenti di stampa globali simili ai contesti di stampa che si trovano sotto Windows, e tutte cose che rendono possibile ad utenti più avanzati di far interagire e collaborare le applicazioni in modi interessanti.

Secondo la concezione Unix di "ambiente desktop" si possono avere programmi di un ambiente che girano in un altro. È concepibile che io possa usare Konqueror sotto GNOME, o Gnumeric sotto KDE. Dopo tutto sono solo programmi. Ovviamente l'idea di ogni ambiente desktop è la consistenza, quindi ha senso usare solo applicazioni progettate per il proprio particolare ambiente; tuttavia se l'utente vuole avere a che fare con un'applicazione che sembra "fuori luogo" e non interagisce bene con il resto dell'ambiente, liberissimo di farlo.

11. Una giornata nei panni di un sistema X

Questo è un esempio di come procede una tipica sessione GNOME in un ambiente desktop moderno su un sistema Linux. Le cose funzionano in maniera simile ad altri ambienti, fermo restando che il sistema alla base sia X.

Quando un sistema Linux fa partire X, il server X si avvia e inizializza il dispositivo grafico, aspettando richieste dai client. Per primo parte un programma chiamato gnome-session e inizializza la sessione di lavoro. Una sessione include cose come applicazioni usate di frequente, la loro posizione su schermo e cose simili. Poi viene avviato il pannello. Il pannello appare in fondo (di solito) ed è una specie di cruscotto per un ambiente a finestre. Permetterà di lanciare programmi, vedere quali sono in esecuzione e di controllare l'ambiente di lavoro. Poi si avvia il window manager. Visto che stiamo usando GNOME potrebbe essere uno dei tanti, ma in questo caso assumiamo che sia Sawfish. Infine viene avviato il file manager (gmc o Nautilus). Il file manager gestisce la presentazione delle icone del desktop. A questo punto l'ambiente GNOME è pronto.

Fin qui tutti i programmi che sono stati avviati sono client, che si connettono al server X. Nel caso specifico il server X risiede sullo stesso computer, ma come abbiamo visto prima ciò non è necessario.

Apriamo un terminale xterm per scrivere qualche comando. Quando clicchiamo sull'icona xterm, il pannello fa partire l'applicazione xterm. È un'altra applicazione client X, quindi essa si avvia, si connette al server X e comincia a visualizzare i suoi elementi. Quando il server X assegna lo spazio su schermo ad xterm, fa in modo che il window manager (Sawfish) decori la finestra con una bella barra del titolo e decida dove essa debba apparire sullo schermo.

Ora navighiamo un po'. Clicchiamo sull'icona di Netscape sul pannello e parte un browser. Bisogna ricordare che questo browser non usa le librerie di GNOME né il toolkit Gtk. Sembra un po' fuori luogo... inoltre non interagisce in maniera molto fluida con il resto dell'ambiente. Ora apriamo il menu "File". Motif fornisce i controlli che appaiono sullo schermo, quindi è compito della libreria di Motif quello di usare la libreria Xlib per disegnare su schermo i componenti necessari a visualizzare il menu e fare in modo che si possa selezionare l'opzione "exit" per chiudere l'applicazione.

Ora apriamo un foglio di calcolo Gnumeric e incominciamo a lavorarci. Ad un certo punto abbiamo bisogno di usare xterm, quindi ci clicchiamo sopra. Sawfish vede questa azione ed essendo incaricato della gestione delle finestre porta xterm in primo piano e gli concede il focus di modo che ci si possa lavorare.

Dopodiché, torniamo al nostro foglio di calcolo; avendo finito di modificarlo vorremmo stampare il nostro documento. Gnumeric è un'applicazione GNOME, quindi può usare gli strumenti forniti dall'ambiente GNOME. Quando stampiamo Gnumeric richiama la libreria gnome-print, che comunica effettivamente con la stampante e produce la copia su carta di cui ho bisogno.

12. Copyright and License