Ghid de utilizare UNIX:

Interfața grafică în UNIX/Linux

Cristian Vidrașcu

Sumar

- Introducere
- Modelul X Window System
- Managerul de ferestre
- Viitorul model: Wayland

Introducere

- **X Window System** – modulul responsabil cu afișarea interfeței grafice în majoritatea sistemelor Unix/Linux (este o specificație, nu un program)

  X – succesorul lui *W Window System*

  Window – elementul central în orice interfață grafică

  System – format din mai multe componente

- Specificație independentă de platforma hardware
- Istoric: proiect început în 1984 la MIT
- Coordonator actual: Fundația X Org

Introducere

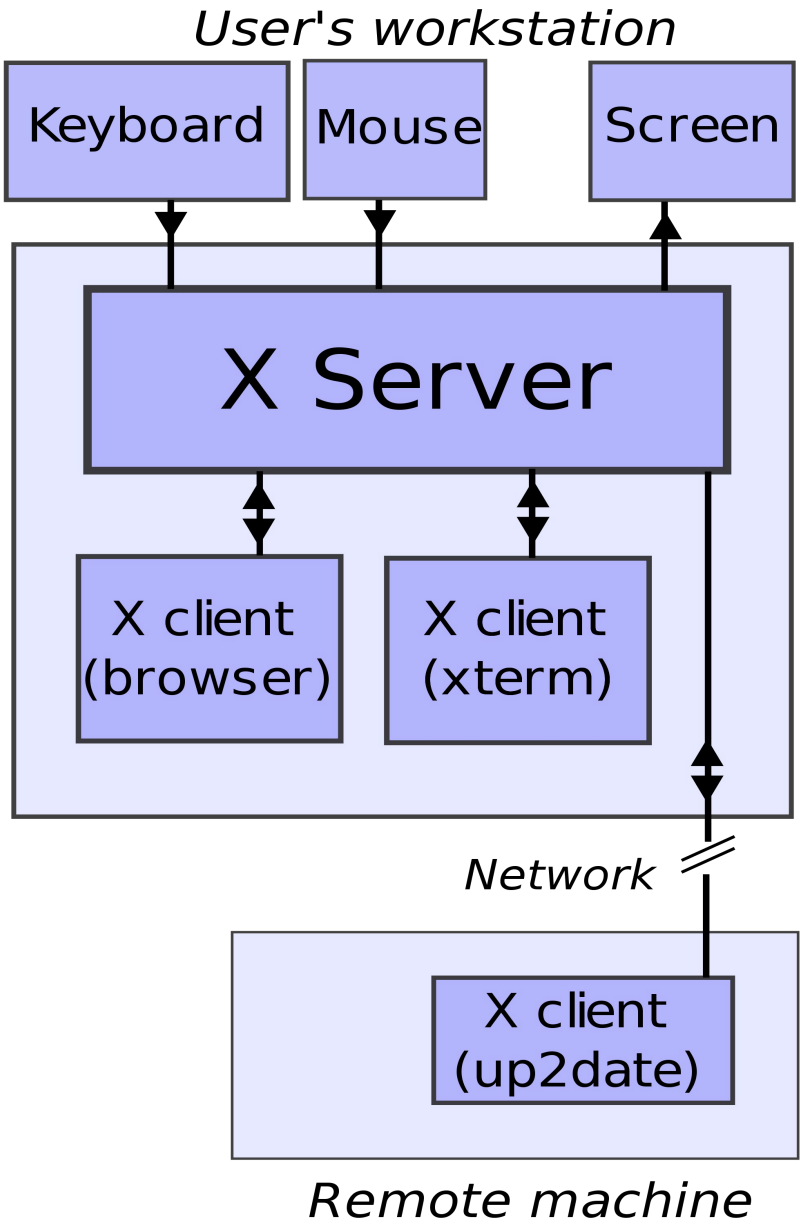
- Implementarea de referință – Xfree86 / X.Org
- Versiunea curentă: X11R7.x
- Istoricul versiunilor: vezi pe www.x.org
- Există și alte implementări comerciale sau gratuite, inclusiv pentru Microsoft Windows
- X nu este singurul model de interfață grafică pentru sisteme Unix, dar e singurul larg acceptat
  Alte modele: W Window System (precursorul lui X), NeWS (Network extensible Window System), NeXT Display PostScript, Berlin/Fresco, Y Window System, ș.a.

Modelul X Window System

Concepte

- *Consolă* – este compusă în general din ecran, tastatură și mouse
- *Drivere* – pentru comunicarea cu perifericele
- *Server X* – proces responsabil cu managementul ferestrelor, fonturilor și a cursorului
- *Client X* – aplicație ce utilizează ferestre gestionate de serverul X
- *Protocol* – set de reguli pentru comunicația între clienți și server
- *Manager* – administrator de ferestre
- *Toolkit* – bibliotecă de componente grafice
- *Terminal X* – terminal cu un server X instalat, ce comunică cu clienți aflați, eventual, pe alte sisteme

Client – Server



- Clientul se conectează la server, cerând afișarea de ferestre (conectare prin canale locale sau socket-uri)
- Serverul procesează cererile clienților și afișează rezultatele pe ecran
- Evenimentele de la tastatură și mouse sunt notificate clienților

Serverul X

- Rol de intermediar între programe și dispozitivele hardware de comunicare cu utilizatorul
- Suport pentru mai multe aplicații client
- Suport pentru mai multe console
- Gestionează fonturi – XFontServer
- Inițializat cu comanda xinit
- Nu oferă componente avansate de interacțiune
- Oferă suport pentru comunicarea între aplicații client (clipboard comun, drag&drop între aplicații, ș.a.)

Aplicații X (clienți X)

- Client X = orice aplicație care utilizează protocolul X pentru a comunica cu un server X
- Nu utilizează resursele hardware pentru a-și crea interfața grafică, ci apelează metodele serverului grafic X
- Nu se impun restricții asupra limbajului de programare, sistemului de operare sau a platformei hardware
- Se poate afla pe alt calculator decât cel pe care se află serverul X

Pornirea serverului X

- xinit

  Pornește serverul X și procesează fișierul ~/.xinitrc

  Dacă are ca parametru calea către un program, execută acest program.

  E utilizat de alte modalități de a porni serverul X ca punct de pornire:

- startx

  Pornește serverul X și un window manager pentru utilizatorul curent

- init 5

  Pornește serverul X și un window manager cu fereastră de login

Console text și grafice

Linux este un S.O. multi-user - permite simultan lucrul cu:

- 6 console text, locale (i.e. 6 sesiuni de lucru în mod text)

  Se comută între ele cu combinația de taste Ctrl+Alt+F1,...,Ctrl+Alt+F6.

  Terminalele sunt referite prin numele tty1,...,tty6.

- până la 6 console grafice, locale

  Se comută între ele cu combinația de taste Ctrl+Alt+F7,...,Ctrl+Alt+F12.

  Trebuie pornit câte un server X în fiecare dintre ele:

  `startx -- :0` (sau, pe scurt, `startx`)

  `startx -- :1`

  ...

  `startx -- :5`

Notă: în unele distribuții, a 6-a consolă grafică este folosită de kernel pentru afișarea continuă a mesajelor de stare, de eroare, etc.

Elemente grafice de bază: Ferestrele

- Singurul element prin care sunt create interfețe
- Orice componentă de interfață este creată utilizând un număr de ferestre
- Organizate ierarhic într-o structură arborescentă
- Există o singură fereastră rădăcină, “the root window”
- Pot exista și servere X fără fereastră rădăcină – “rootless server”, de exemplu în cazul în care acel X nu este serverul grafic principal, ci este pornit ca aplicație într-un alt mediu vizual (pentru rularea de aplicații X în medii MacOS X sau Windows)

Alte elemente de bază: Evenimentele

- Evenimente = mesaje trimise de server clientului pentru a notifica faptul că a survenit ceva ce ar putea interesa aplicația
- Evenimente de la dispozitivele de intrare – mouse, tastatură
- Evenimente vizuale – ascunderea sau reafișarea unei ferestre
- Evenimente trimise de o aplicație către o fereastră a altei aplicații – comunicare inter-aplicații
- O aplicație specifică tipurile de evenimente ce dorește să-i fie notificate
- Modelul de programare *event-driven*

Managerul de ferestre

- Este un client X special
- Responsabil cu afișarea “decorațiilor” ferestrelor (bara de titlu, butoanele sistem, marginile de redimensionare, etc.)
- Responsabil cu oferirea unui *desktop* (spatiu de lucru) și cu gestionarea aplicațiilor active
- Nu poate să fie activ decât un singur manager de ferestre la un moment dat

Managerul de ferestre

- Clasificare după modul de desenare și actualizare a ferestrelor pe ecran:
  - *stacking window manager* – ferestre suprapuse
  - *tiling window manager* – ferestre alăturate
  - *compositing window manager* – efecte vizuale avansate 2D și 3D (e.g. Compiz/Beryl)

Managerul de ferestre

- Există o gamă variată de managere de ferestre, de la unele minimale la adevărate suite de aplicații

- Clasificare după complexitate:

  - *managere de ferestre* (simple): controlează plasarea și aspectul ferestrelor în cadrul interfeței grafice

  - *medii desktop*: includ un manager de ferestre, plus un manager de fișiere, un set de teme vizuale (*visual themes*) și programe pentru administrarea desktop-ului

Managerul de ferestre

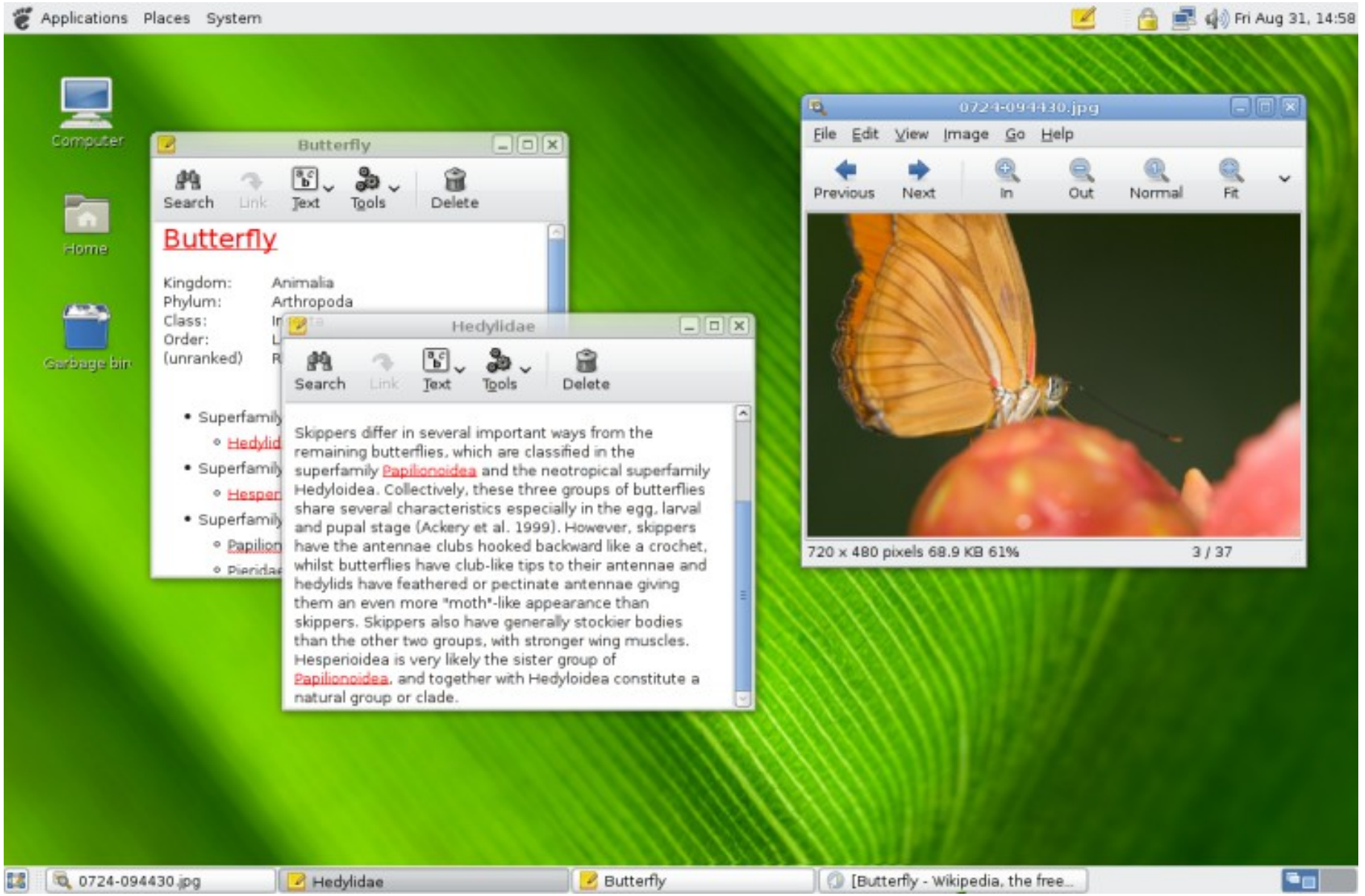
- Cele mai utilizate medii desktop:
  - GNOME
  - KDE
- Alte exemple: Xfce, Enlightenment, CDE, ș.a.

- Unele medii desktop sunt configurabile în privința componentelor pe care le folosesc

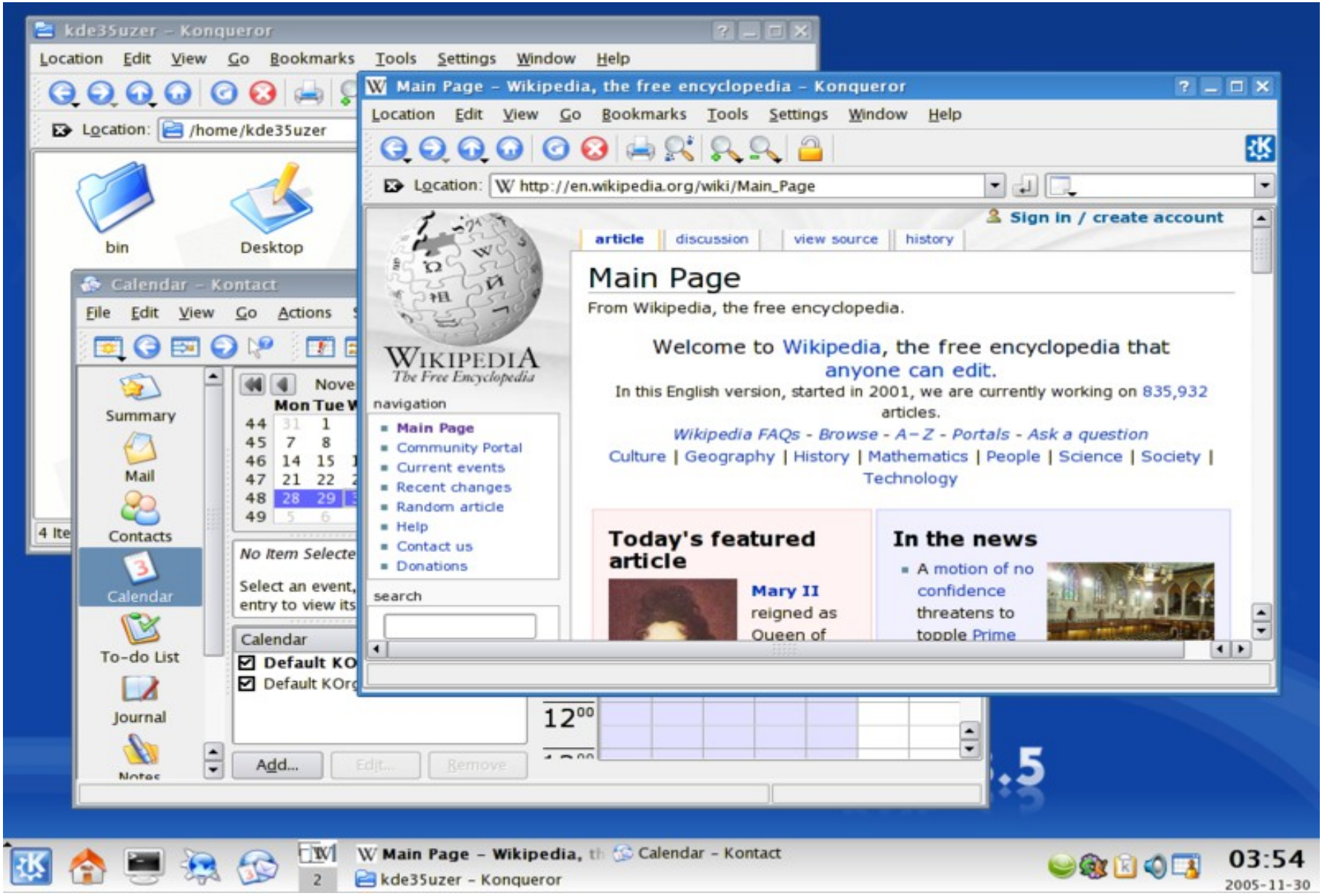
  Exemplu: în GNOME se poate configura folosirea Compiz-ului drept manager de ferestre în locul Metacity-ului (i.e. cel implicit, de tip stacking window manager).

  Similar, în mediul desktop KDE putem folosi Kwin sau Compiz drept compositing windows manager în locul celui clasic.

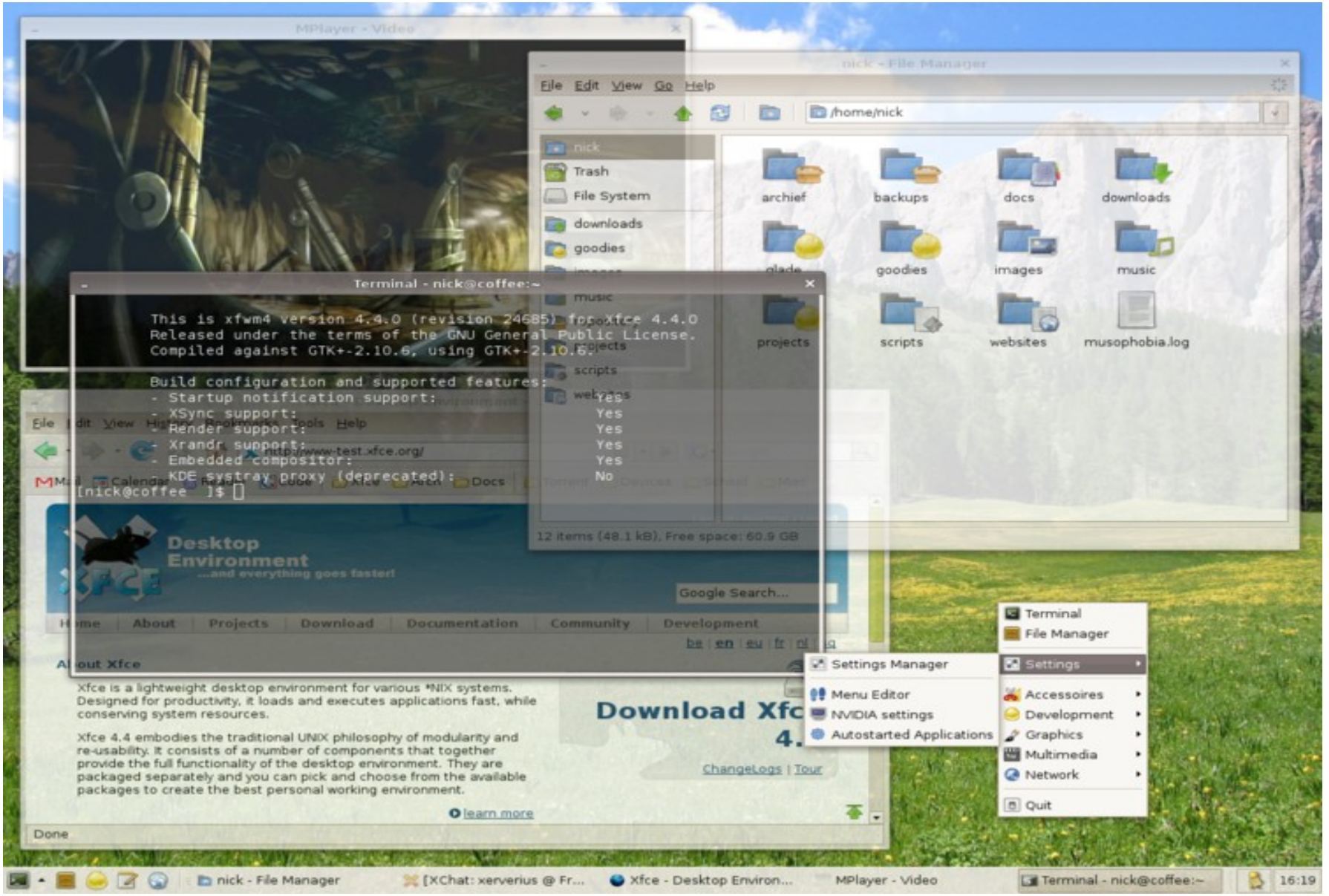
Screenshot – GNOME 2.20 cu Metacity



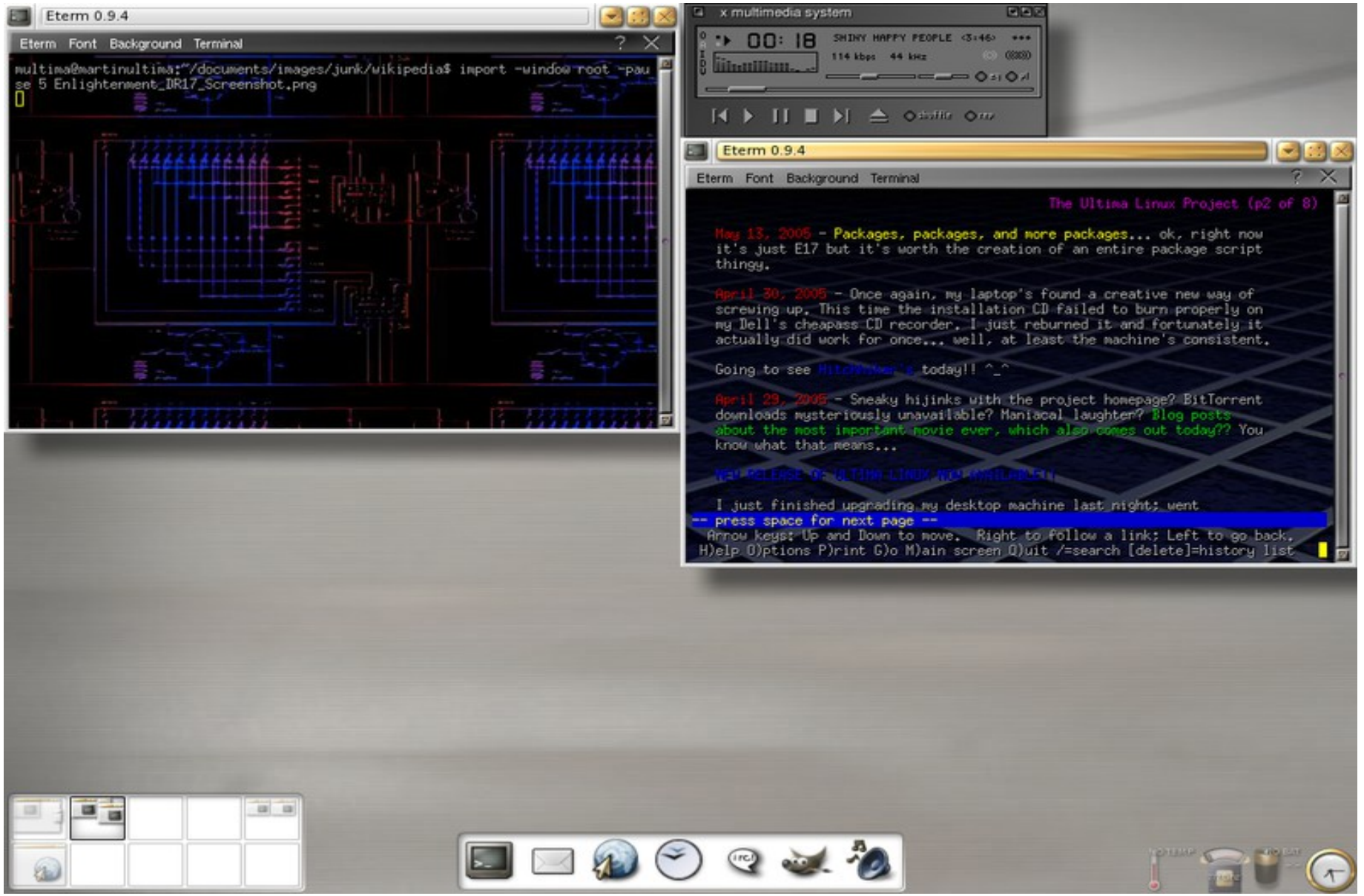
Screenshot – KDE 3.5



Screenshot – Xfce



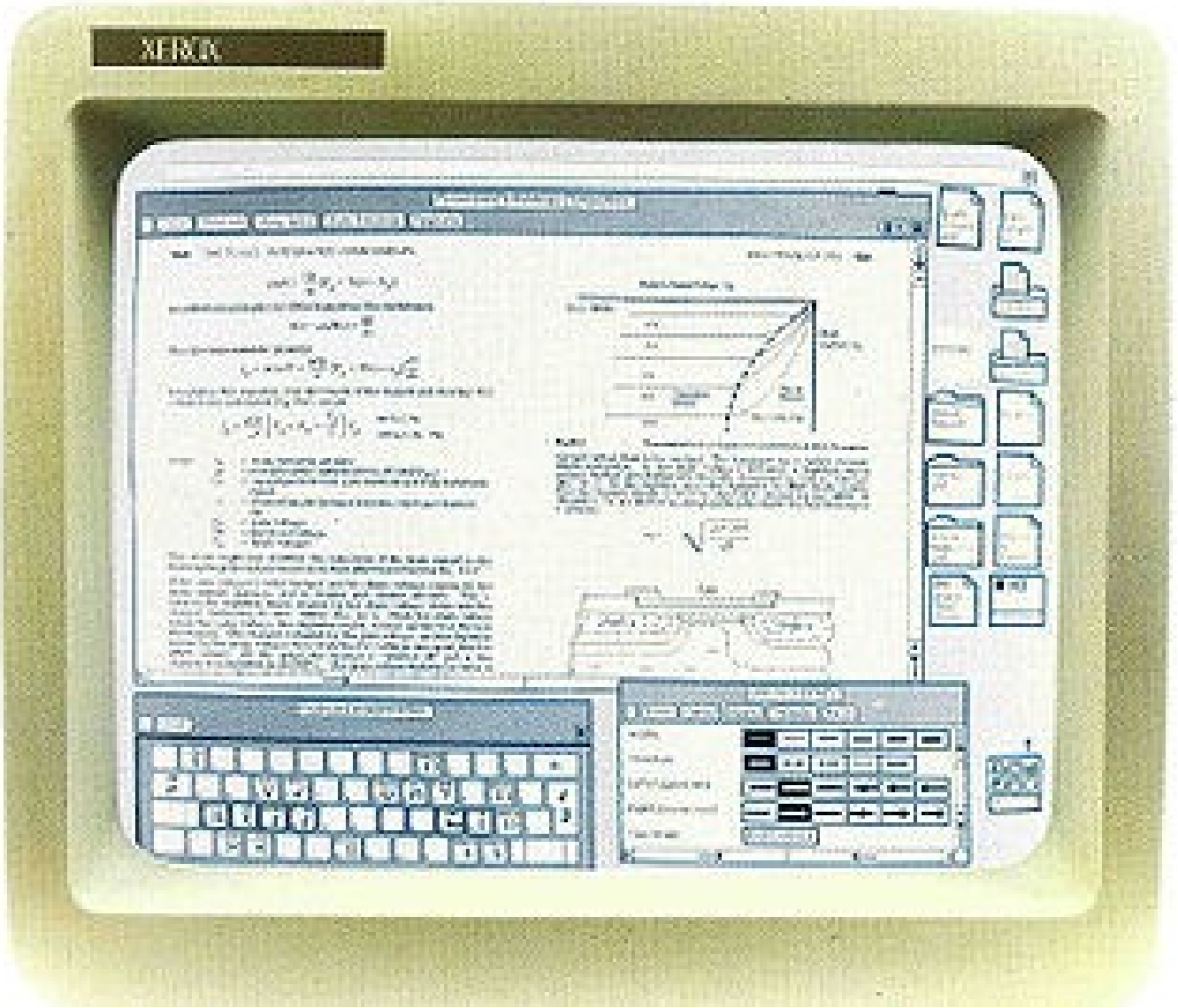
Screenshot – Enlightenment



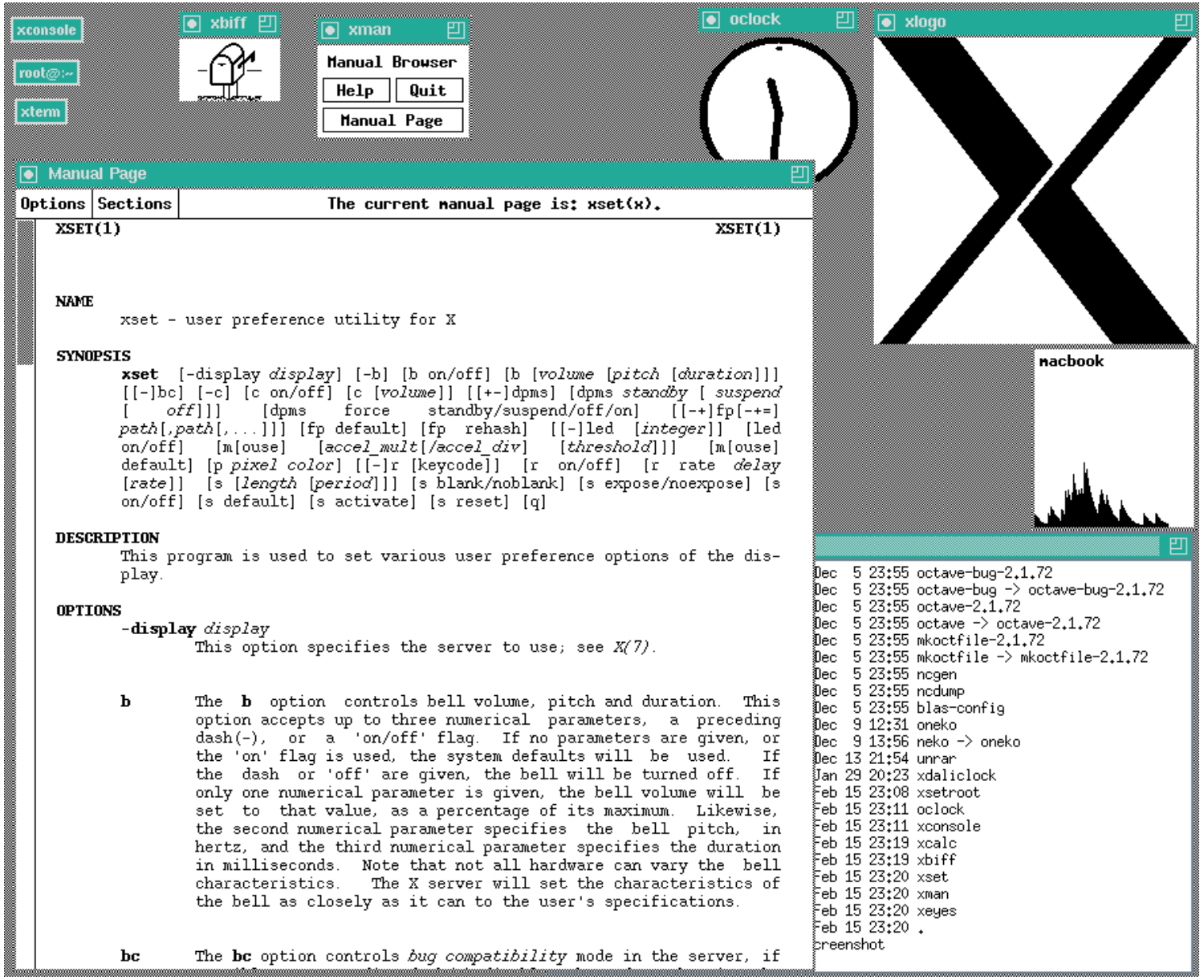
Istorie: primul mouse @SRI

(Stanford Research Institute)

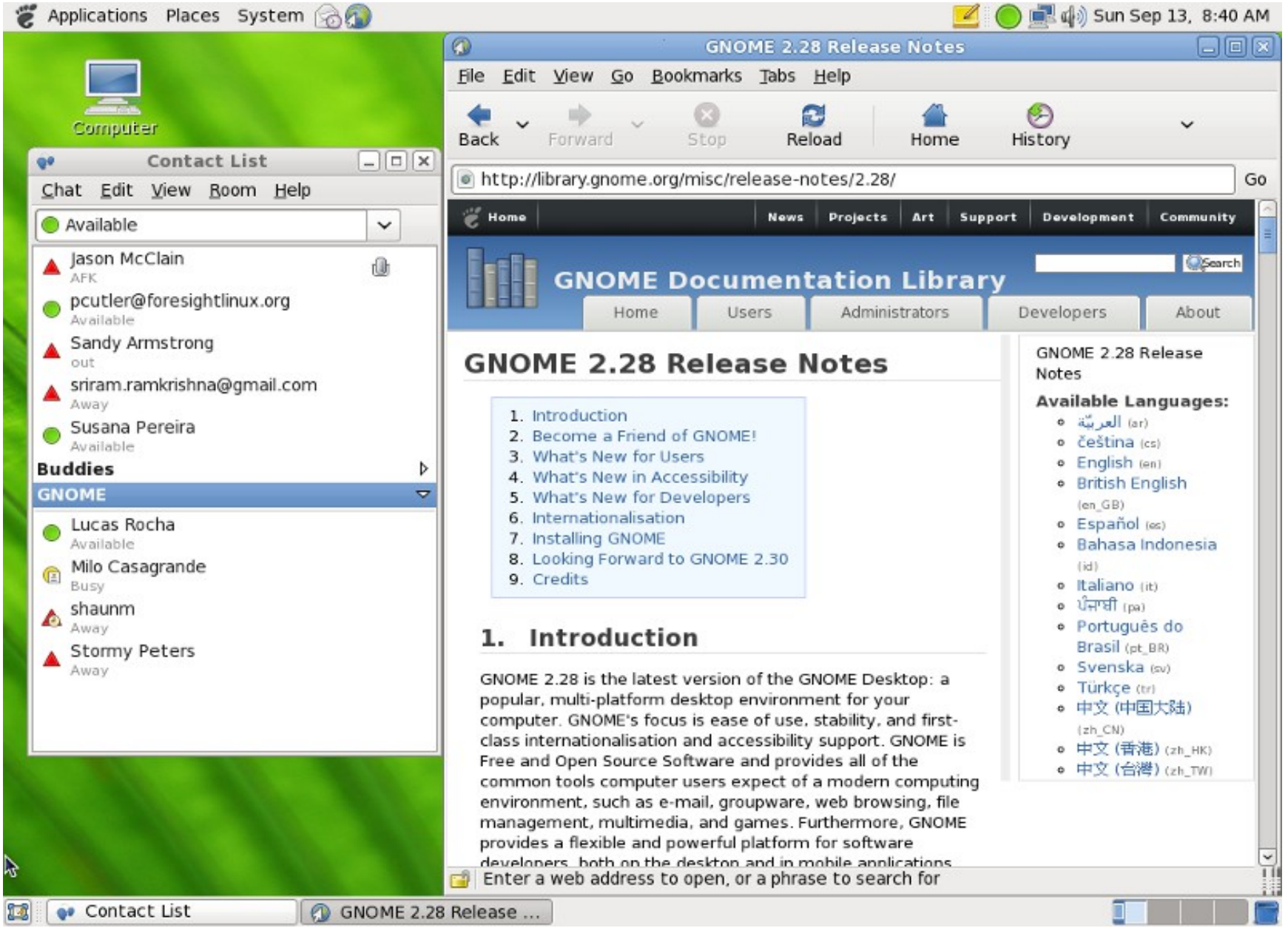
Istorie: primul GUI @ Xerox PARC



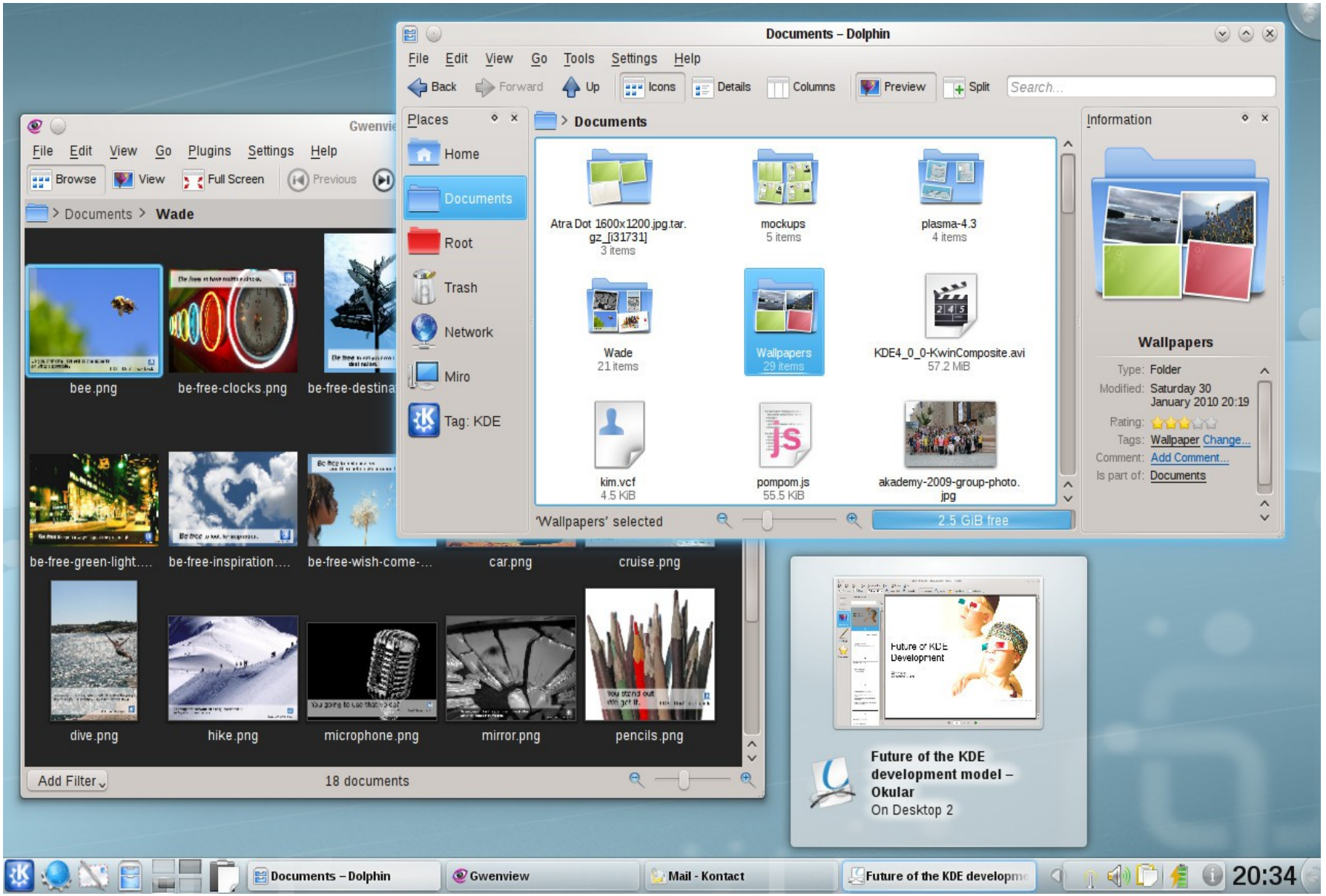
Istorie: A Unix based X Window System desktop (circa 1990)



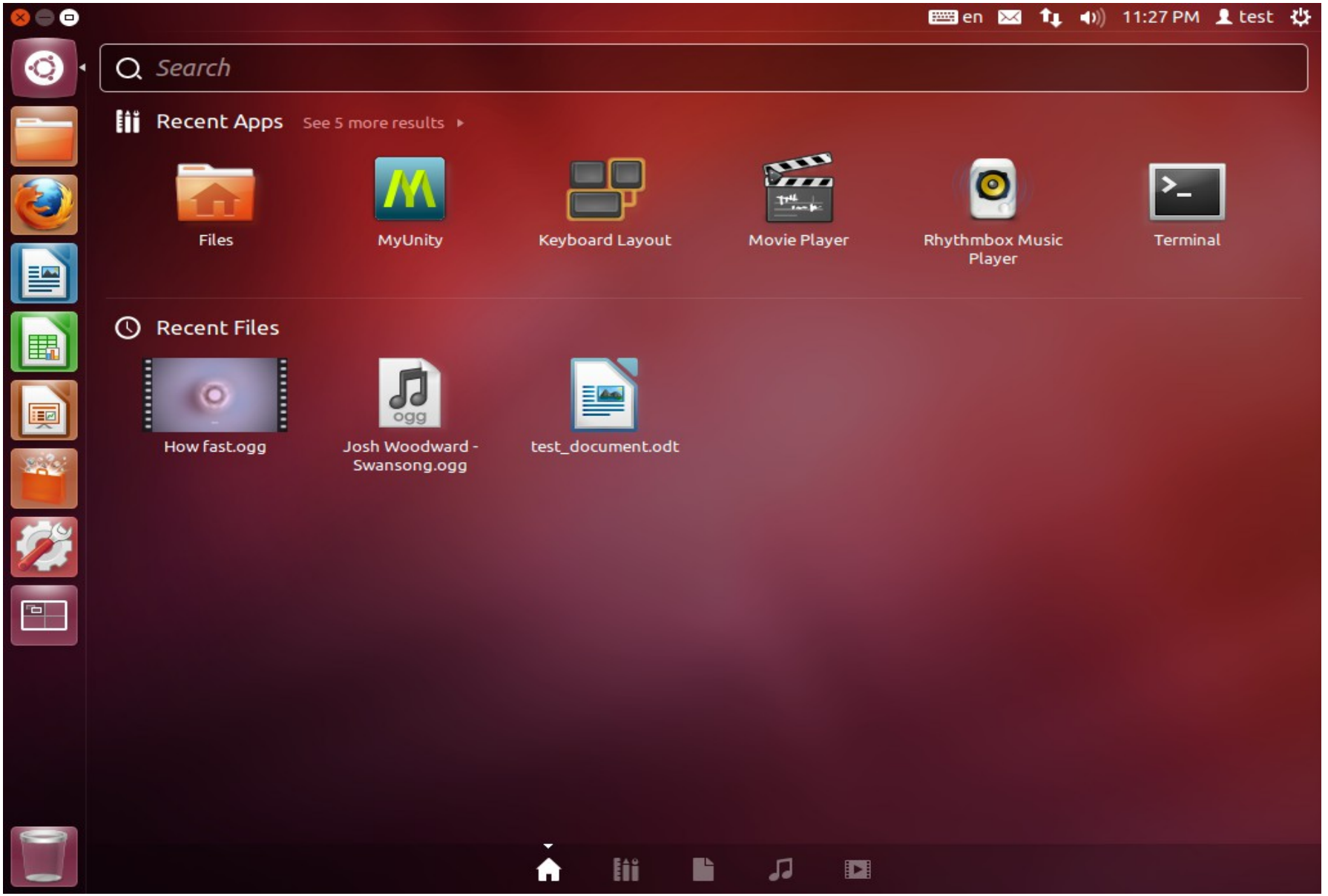
Prezent: GNOME 2.28 desktop (2010)



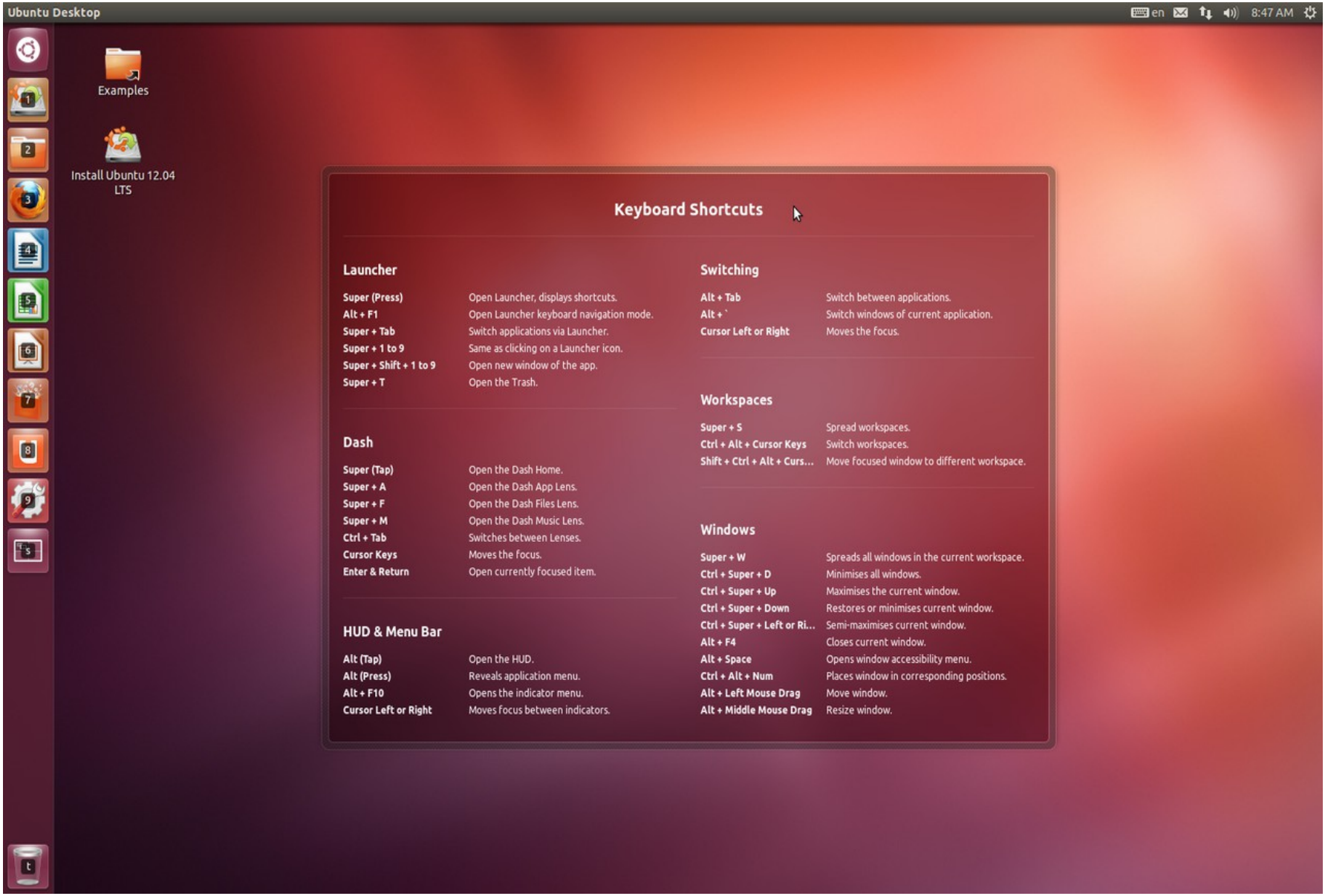
Prezent: KDE Plasma 4.4 desktop (2010)



Prezent: Unity - shell interface în Ubuntu 2012.04 (2012)



Prezent: Unity - shell interface în Ubuntu 2012.04 (2012)



3D GUI

- Compositing window manager

  - un manager de ferestre care furnizează aplicațiilor un buffer off-screen pentru a memora ferestrele aplicației, apoi compune bufferele tuturor aplicațiilor într-o singură imagine și o copie în memoria plăcii grafice

  - în plus, procesează suplimentar bufferele cu ferestrele aplicațiilor, aplicând variate efecte vizuale/animații 2D și 3D precum ar fi: blending, fading, scaling, rotation, duplication, bending and contortion, shuffling, blurring, etc.

3D GUI

- Compiz = a compositing window manager for Linux

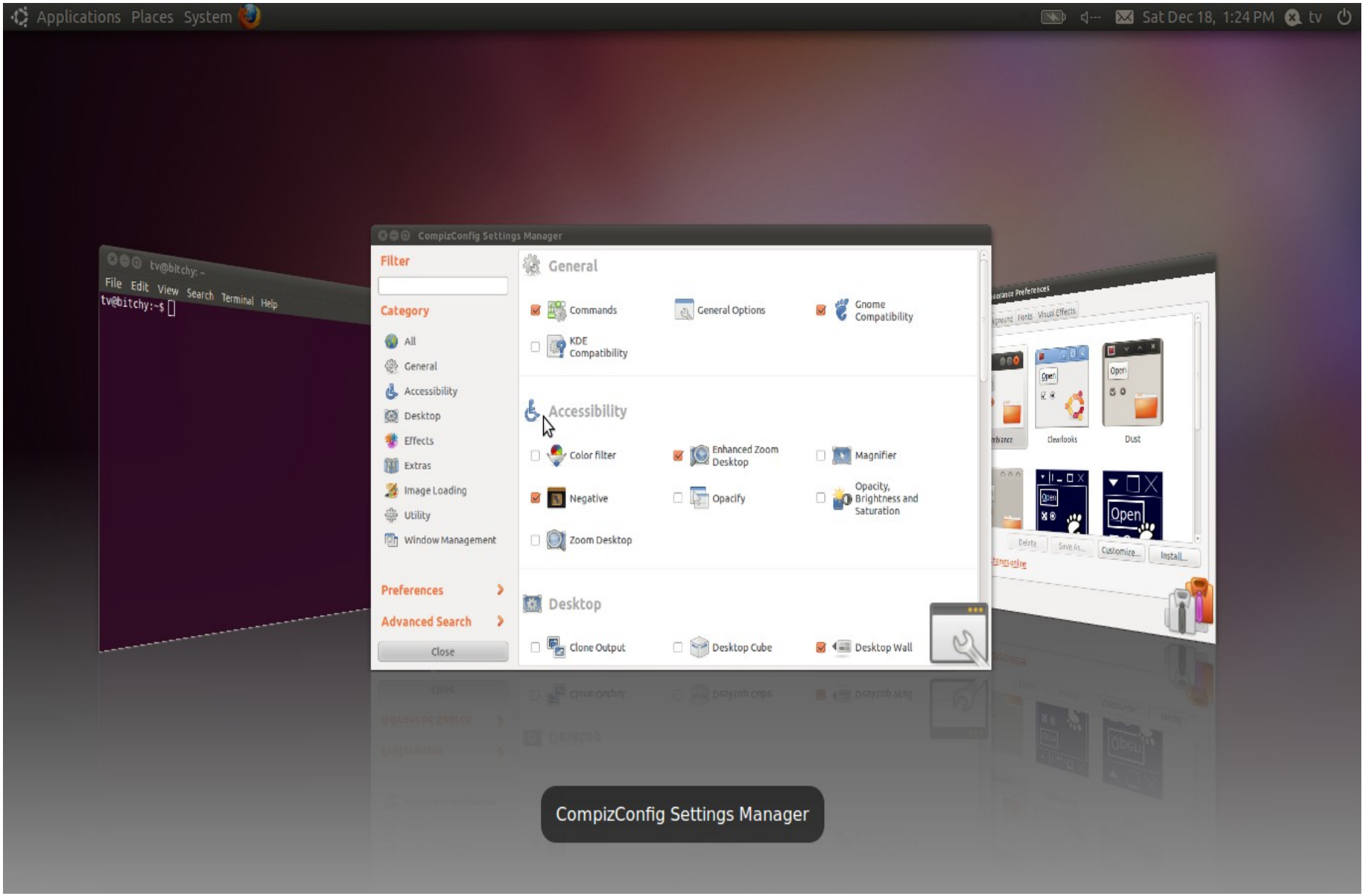
  - Instalare în Ubuntu: pachetele compiz și compiz-extra-plugins

  - Plugin-urile Compiz includ efecte speciale, precum ar fi:

    - the desktop cube effect
    - Alt-Tab application-switching with live previews or icons
    - efectul Exposé (the scale plugin în Compiz)
    - the fire effect, the rain effect, ș.a.

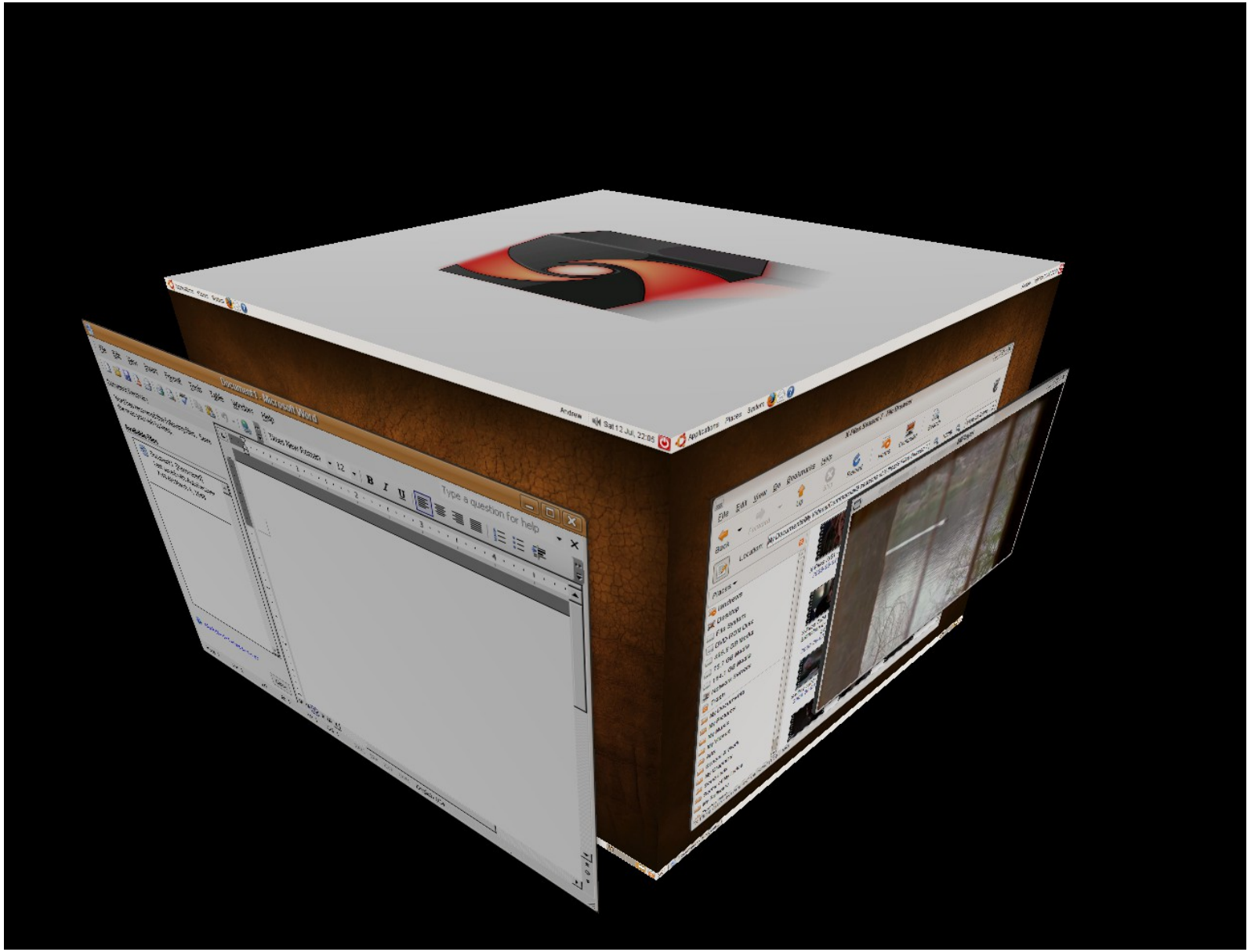
  - Plugin-urile Compiz sunt clasificate în 4 grupe: Main, Extra, Unsupported și Experimental

Compiz: panoul de configurare a efectelor 3D



Compiz running on Fedora Core 6 with AIGLX
(the desktop cube effect)

Compiz+GNOME running on Ubuntu
(the desktop cube effect)



Kwin 4.4 in KDE
(the desktop cube effect)

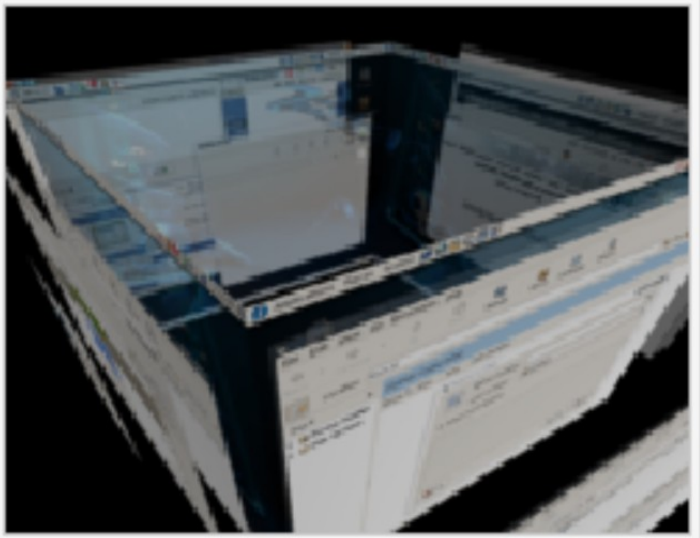
Compiz: the screen magnifier effect

article | discussion | edit this page | history | move | unwatch

Compositing window manager

From Wikipedia, the free encyclopedia

A **compositing window manager** is a component of a computer's graphical user interface that draws windows and/or their borders. It also controls how they are displayed and interact with each other, and the rest of the desktop environment. The main difference between compositing window manager and a standard window manager is that instead of outputting to a common screen, programs each output first to a separate and independent buffer, or temporary location inside the computer, where they can be manipulated before they are shown. [1] [2]

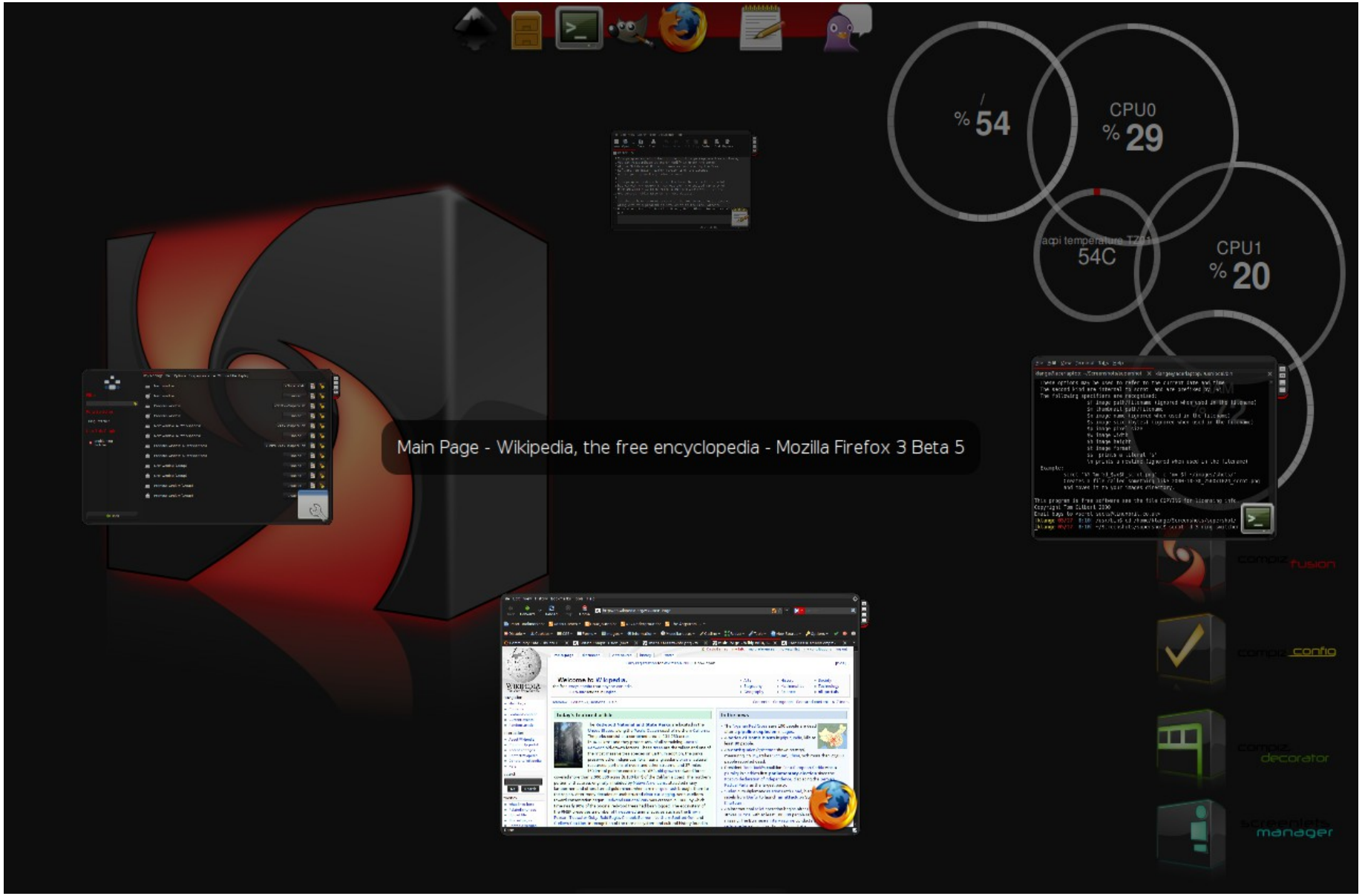


Compiz, a compositing window manager, running on Fedora Core 6

Compiz: the shift switcher in flip mode (Ubuntu 8.04)

Kwin: the shift switcher in cover mode

Compiz: the shift switcher in ring mode



Viitorul model

- Wayland – un proiect început în 2008, drept înlocuitor pentru X Window System, dar numai pentru sisteme Linux (folosește anumite funcționalități înglobate în nucleul Linux)

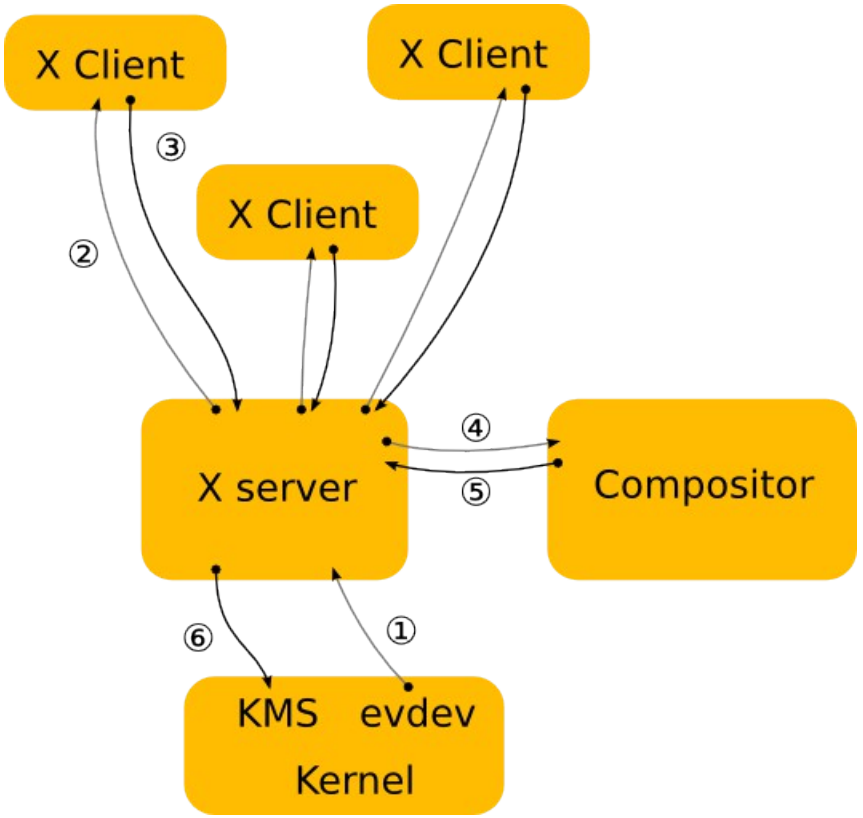
  Wayland : display server protocol și o bibliotecă ce-l implementează

  Weston : o implementare de referință a unui compositor pentru Wayland

- Mir – un proiect anunțat de firma Canonical Ltd. în martie 2013, drept înlocuitor în versiunile viitoare de Ubuntu pentru X Window System, în loc de Wayland

Wayland vs. X Window System

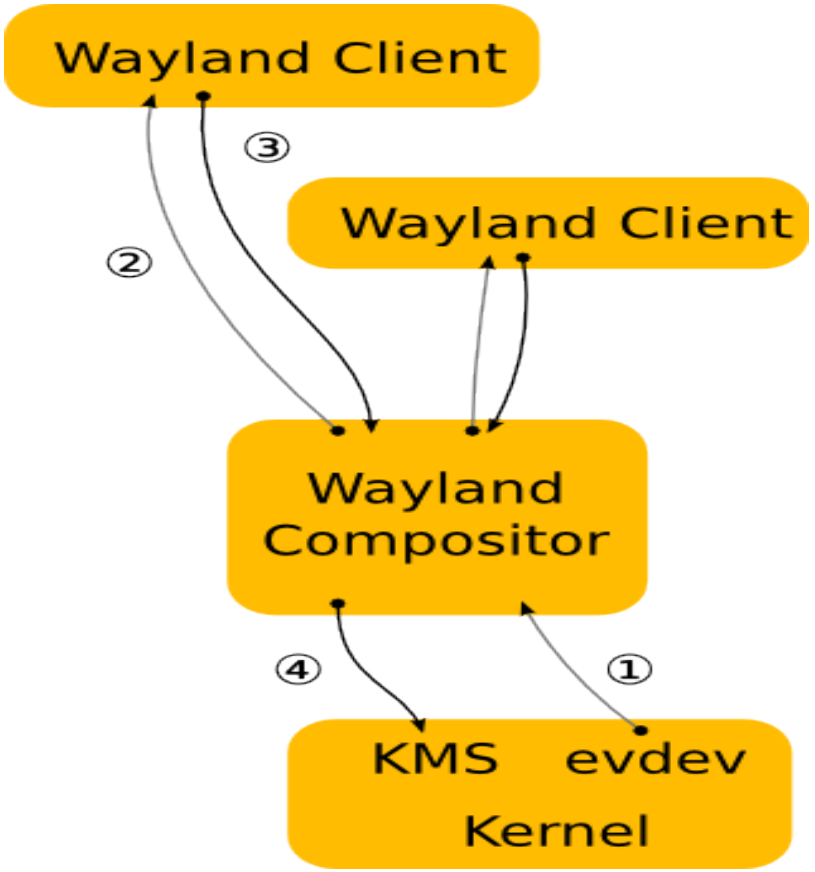
- Arhitectura X Window System



1. The kernel gets an event from an input device and sends it to X through the evdev input driver. The kernel does all the hard work here by driving the device and translating the different device specific event protocols to the linux evdev input event standard.
2. The X server determines which window the event affects and sends it to the clients that have selected for the event in question on that window. The X server doesn't actually know how to do this right, since the window location on screen is controlled by the compositor and may be transformed in a number of ways that the X server doesn't understand (scaled down, rotated, wobbling, etc).
3. The client looks at the event and decides what to do. Often the UI will have to change in response to the event - perhaps a check box was clicked or the pointer entered a button that must be highlighted. Thus the client sends a rendering request back to the X server.
4. When the X server receives the rendering request, it sends it to the driver to let it program the hardware to do the rendering. The X server also calculates the bounding region of the rendering, and sends that to the compositor as a damage event.
5. The damage event tells the compositor that something changed in the window and that it has to recomposite the part of the screen where that window is visible. The compositor is responsible for rendering the entire screen contents based on its scenegraph and the contents of the X windows. Yet, it has to go through the X server to render this.
6. The X server receives the rendering requests from the compositor and either copies the compositor back buffer to the front buffer or does a pageflip. In the general case, the X server has to do this step so it can account for overlapping windows, which may require clipping and determine whether or not it can page flip. However, for a compositor, which is always fullscreen, this is another unnecessary context switch.

Wayland vs. X Window System

- Arhitectura Wayland



1. The kernel gets an event and sends it to the compositor. This is similar to the X case, which is great, since we get to reuse all the input drivers in the kernel.
2. The compositor looks through its scenegraph to determine which window should receive the event. The scenegraph corresponds to what's on screen and the compositor understands the transformations that it may have applied to the elements in the scenegraph. Thus, the compositor can pick the right window and transform the screen coordinates to window local coordinates, by applying the inverse transformations. The types of transformation that can be applied to a window is only restricted to what the compositor can do, as long as it can compute the inverse transformation for the input events.
3. As in the X case, when the client receives the event, it updates the UI in response. But in the wayland case, the rendering happens in the client, and the client just sends a request to the compositor to indicate the region that was updated.
4. The compositor collects damage requests from its clients and then recomposites the screen. The compositor can then directly issue an ioctl to schedule a pageflip with KMS.