Il progetto Utopia

In questo articolo si esaminerà il progetto Utopia nel suo insieme, e più in dettaglio, i suoi componenti. Verranno inoltre citati alcuni software che pur non facenti parte del progetto sono essenziali al funzionamento globale del sistema.

Prima di iniziare sono necessarie alcune precisazioni che possono facilitare la lettura: nel testo che segue il termine dispositivo e il suo equivalente inglese device (plur. devices) sono usati per indicare un dispositivo fisico/periferica. Il termine device è usato anche come abbreviazione di device driver a indicare il programma che pilota la periferica o comunque un modulo software con questo uso.

Come da documentazione ufficiale il progetto utopia si propone come scopo principale la realizzazione di uno strato di interfaccia, quanto più possibile, trasparente all'utente finale in grado di realizzare il plug & play dei dispositivi sui desktop GNU/Linux.

Il progetto, iniziato da Robert Love, si basa su 4 software che funzionano in maniera indipendente ma che possono comunicare e collaborare tra di loro:

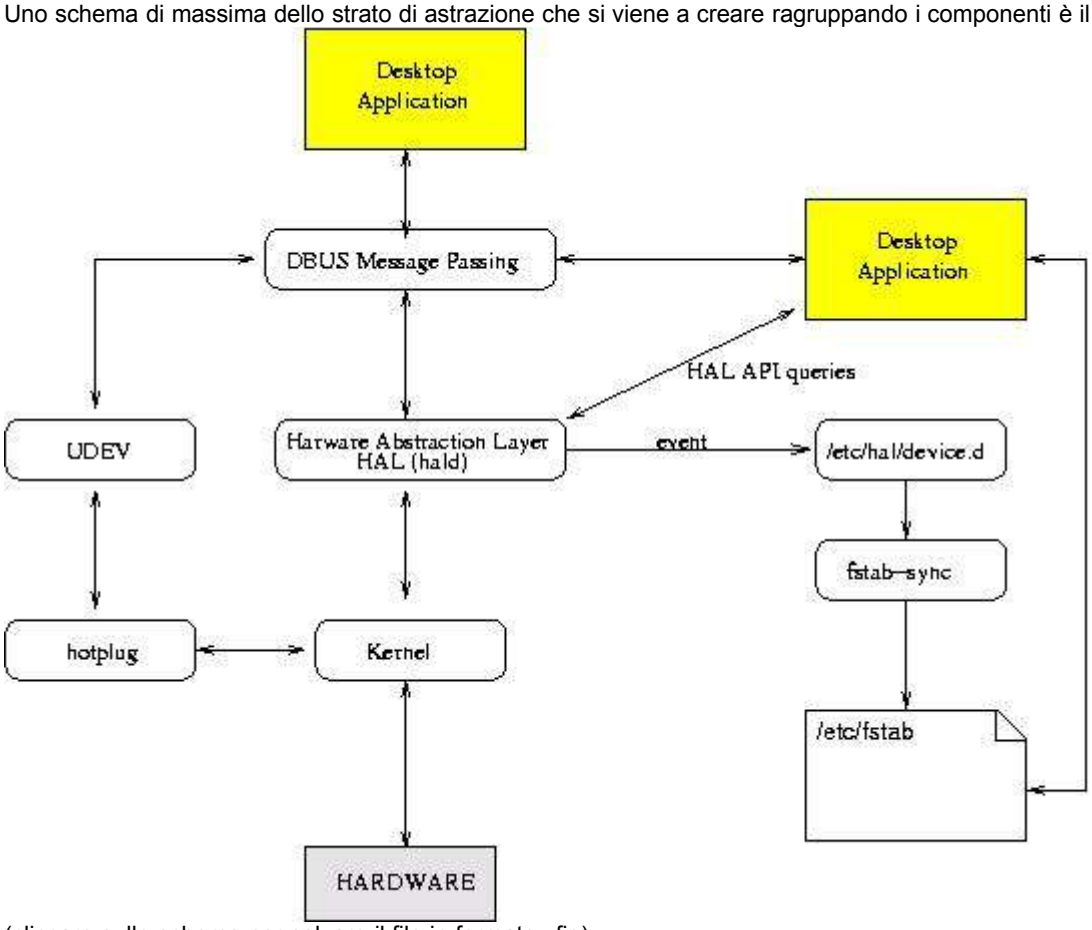
1. HAL [http://www.freedesktop.org/Software/hal] - Hardware Abstraction Layer: provvede l' accesso all' hardware alle applicazioni desktop

2. DBUS [http://www.freedesktop.org/Software/dbus] - fornice il passaggio di messaggi fra le applicazioni

3. UDEV [http://kernel.org/pub/linux/utils/kernel/hotplug/udev.html]- rimpiazza DevFS nella gestione dei device, interagisce con il demone hotplug e il kernel per la gestione dei device.

4. GNOME Volume Manager - Gestisce l' interfaccia utente e le icone dei devices. Non verrà trattato in questa sede.

Uno schema di massima dello strato di astrazione che si viene a creare ragruppando i componenti è il seguente:



(cliccare sullo schema per salvare il file in formato xfig)

Vediamo nel dettaglio i componenti del progetto per meglio comprendere come si legano fra loro.

HAL

È l' ennesimo strato di astrazione per l' hardware (finiranno un giorno gli strati della torta?) creato per fornire alle applicazioni un modo per interagire direttamente con l' hardware.
Per tradizione, e anche per motivi di sicurezza, nei sistemi GNU/Linux (e per quanto mi è dato modo di sapere anche negli altri sistemi *NIX) tale compito è sempre stato demandato al kernel . È pur vero che, tali ragioni di sicurezza, erano molto più valide quando tali sistemi erano condivisi fra più utenti, mentre la prospettiva cambia con gli attuali sistemi desktop che sono, in maggioranza, monoutente e nei quali conta molto l'usabilità. In quest' ottica viene introdotto questo strato che porta la gestione dell' hardware al di fuori del kernel (mi si perdoni la semplificazione introdotta) permettendo una maggiore iterazione con gli applicativi in user space.
Quando un nuovo dispositivo viene aggiunto al sistema un segnale asincrono viene diffuso sul bus dei messaggi di sistema (DBUS); tale segnale contiene il maggior numero possibile di dettagli sul dispositivo stesso, informazioni che vengono così messe a disposizione degli applicativi. Potete osservare questo comportamento aprendo, come root, una console Xterm, eseguendo il comando
tail -f /var/log/messages
ed inserendo una periferica usb come ad esempio un ram key (o flash drive che dir si voglia).
Si ottiene un output come il seguente:

Jan 28 16:27:09 giacomini kernel: usb 1-1: new full speed USB device using address 3
Jan 28 16:27:09 giacomini kernel: scsi1 : SCSI emulation for USB Mass Storage devices
Jan 28 16:27:09 giacomini kernel: Vendor: CONE Model: 128MB i-Disk Rev: 1.89
Jan 28 16:27:09 giacomini kernel: Type: Direct-Access ANSI SCSI revision: 02
Jan 28 16:27:09 giacomini kernel: SCSI device sda: 255744 512-byte hdwr sectors (131 MB)
Jan 28 16:27:09 giacomini kernel: sda: Write Protect is off
Jan 28 16:27:09 giacomini kernel: sda: assuming drive cache: write through
Jan 28 16:27:09 giacomini kernel: sda: unknown partition table
Jan 28 16:27:09 giacomini kernel: Attached scsi removable disk sda at scsi1, channel 0, id 0, lun 0
Jan 28 16:27:09 giacomini scsi.agent[3521]: disk at /devices/pci0000:00/0000:00:02.2/usb1/1-1/1-1:1.0/host1/1:0:0:0
Jan 28 16:27:10 giacomini fstab-sync[3607]: added mount point /media/usbdisk for /dev/sda

Il demone hald (sul quale HAL basa il proprio funzionamento) mantiene una lista interna di oggetti per ciascuno dei quali è memorizzata una copia valore-chiave che lo identifica. Tale chiave è detta UDI (Unique Device Identifier) e serve appunto a hald per recuperare le informazioni sul dispositivo dal kernel e da eventuali fonti aggiuntive (file di configurazione personalizzati ad es.).
Una volta identificato il device vengono avviati i programmi contenuti nella directory /etc/hal/device.d/ fra i quali fstab-sync programma che va a modificare il file /etc/fstab rendendo così disponibili le informazioni su eventuali nuovi mount points anche ai programmi legacy che non sono in grado di ricevere le notifiche dal bus di sistema.
Chiaramente è possibile per l' amministratore di sistema modificare sia i mount points sia l'acessibilità ad essi da parte degli utenti cosa che si può ottenere tramite un file (XML) di configurazione posizionato in /usr/share/hal/fdi/95userpolicy/ per la sintassi del quale Vi rimando alla documentazione ufficiale.
Potete avere un elenco completo delle periferiche gestite da hald nel vostro sistema con il comando
lshal | more
da tale lista è possibile dedurre un grande numero di informazioni sui dispositivi e su come vengono "visti" dal sistema. Fra le varie informazioni potete osservare le righe che iniziano con info.udi che contengono gli identificativi (UDI) dei dispositivi.
Vi sono poi altri due comandi nel pacchetto: hal-get-propety e hal-set-property che possono essere utilizzati (negli script) per ottenere o settare le proprietà del device.

Nota:
fstab-sync - Aggiorna il file /etc/fstab in risposta ad eventi segnalati da hald (Il file fstab contiene informazioni descrittive sui vari file system; normalmente fstab è; solamente letto dai programmi, e non scritto) e crea/rimuove i mount points nella directory /media.Il programma viene invocato tramite chiamate esterne, ai file della directory /etc/hal/device.d, da HALD.
È facile osservare che, quando il file /etc/fstab è scritto/modificato da fstab-sync, viene aggiunto in testa ad esso un commento relativo.

DBUS

È un sistema per la comunicazione interprocesso (IPC) composto da una libreria (libdbus), un demone (messagebus), e delle "wrapper libraries" ovvero delle librerie di interfaccia per i singoli framework di sviluppo (es. libdbus-glib, libdbus-qt).
Nel tempo ci sono state decine di sistemi IPC sviluppati negli ambienti *NIX e anche IPC multipiattaforma. La caratteristica principale di DBUS, che ne giustifica l' implementazione a scapito dell' utilizzo di uno dei sistemi esistenti, è che esso è stato progettato per il passaggio di messaggi fra applicazioni desktop ed è stato ottimizzato per avere una bassa latenza e un basso overhead, in modo da non portare carico aggiuntivo sul sistema. L' architettura di DBUS si basa su due bus:
il system bus che è unico per tutte le sessioni aperte nel sistema e rappresenta il punto di riferimento sia per il kernel che per gli applicativi in user-space.
Il session bus è invece un bus locale creato per la sessione aperta e per lo specifico utente ed è usato per lo scambio di

messaggi fra applicazioni all' interno della stessa sessione del server X.
Nella progettazione di questo sistema si è posta particolare attenzione a renderlo compatibile, e facilmente adottabile a livello di programmazione, sia dal desktop manager GNOME che da KDE. Per cui si può prevedere che verrà largamente adottato dai programmatori risolvendo il problema dello start up del progetto. Nella fase iniziale, infatti, un sistema del genere ha bisogno che un gran numero di applicazioni lo includano in modo da divenire uno standard. In proposito bisogna notare che DBUS non invia segnali direttamente alle applicazioni, ogni applicazione per utilizzarlo deve infatti registrare gli oggetti (dell' applicazione stessa) che devono ricevere i segnali da DBUS. Alla chiusura dell' applicazione DBUS si incaricherà di terminare ogni servizio relativo all' applicativo e si incaricherà di informare gli altri applicativi della chiusura in questione.

udev

Udev crea (o rimuove) dinamicamente una directory di device drivers, solitamente /dev, in modo che essa contenga solamente i file relativi ai devices effettivamente presenti.

In un sistema GNU/Linux le informazioni relative ai device sono sempre state immagazzinate nella directory /dev popolandola con un nodo (file) per ogni dispositivo che potesse essere teoricamente connesso al computer. Questo sistema, detto devfs, ha portato ad una proliferazione di nodi tale da portare la directory /dev ad essere popolata (secondo le fonti) da oltre 18.000 file. Questo sistema ha mostrato negli ultimi tempi i propri limiti per cui vari programmatori hanno cercato di affrontare la cosa. In udev and devfs - The final word, Greg Kroah-Hartman sostiene in maniera definitiva udev nei confronti del file-system devfs partendo dai seguenti problemi:
1) la directory /dev è inutilmente grande in quanto contiene anche i dispositivi che non avremo mai nel nostro PC
R) udev popola la directory con i soli device dell' harware effettivamente presente
2) I major e minor numbers ovvero i numeri che identificano i vari devices sono in via di esaurimento
R) udev non utilizza i major e minor numbers
3) I device hanno nomi non intuitivi e non sono rinominabili a discrezione dell'utente
R) udev permette di nominare a discrezione dell'utente un device
4) I programmi in user-space non possono ricevere notifica dinamica delle modifiche ai devices
R) poichè udev genera dei messaggi D-BUS qualsiasi programma in userspace (come ad esempio HAL) può prendere atto di modifiche ai devices

Come si evince dalla pagina man udev fa parte del sistema hotplug per cui viene eseguito ogni qualvolta un kernel-device (dispositivo pilotato dal kernel) o network-device (schede di rete e affini) viene aggiunto o rimosso dal sistema. In particolare udev esegue i programmi con estensione .dev posizionati nella directory /dev.d, procede leggendo la directory /sys (sysfs) in modo da raccogliere informazioni sul device. Tali informazioni vengono usate come chiavi per ottenere un nome univoco per il device ricavato da un database dei devices. In base alle informazioni del database e alle regole di configurazione eventualmente specificate dagli utenti (/etc/udev/rules.d/) vengono creati i corretti device in /dev
La documentazione Red Hat consiglia di non modificare i file di configurazione esistenti ma di aggiungere in /etc/udev/rules.d/ il proprio file nominandolo con un numero basso es. 10-miofile.rules in modo che venga letto prima del file delle regole predefinite /etc/udev/rules.d/50-udev.rules
La sintassi di base per una regola è la seguente:
Key, [Key,...] name [,symlink]
per la comprensione di tale forma e delle regole relative vi rimando all' ottimo documento Writing udev rules di Daniel Drake. Faccio solo un accenno alle possibilità offerte da udev: inserendo nel file di cui si accenava sopra la riga KERNEL="sda", name="chiavetta" si ottiene in /var/log/messages il seguente output:

Jan 28 16:29:42 giacomini kernel: Attached scsi removable disk sda at scsi2, channel 0, id 0, lun 0
Jan 28 16:29:42 giacomini scsi.agent[3739]: disk at /devices/pci0000:00/0000:00:02.2/usb1/1-1/1-1:1.0/host2/2:0:0:
Jan 28 16:29:43 giacomini fstab-sync[3825]: added mount point /media/usbdisk for /dev/chiavetta

Da cui si deduce che è stato creato un device con un nome personalizzato di nome "chiavetta" in /dev ed è stato aggiunto il mount point relativo in /media/usbdisk. Lascio alla vostra fantasia le considerazioni sulle possibilità offerte mentre voglio ribadire che ciò permette di personalizzare anche il "device node" ovvero di poter mantenere sempre lo stesso nome di periferica indipendentemente dalla sua posizione: ad esempio un disco "hda" potrebbe essere sempre denominato "hda" indipendentemente dalla configurazione hardware. La cosa diminuirebbe parecchio la confusione per i nuovi utenti (anche se a mio parere si rischia di crearne molta di più negli utenti esperti o nei vecchi sysadmin...).
Il file di configurazione principale è /etc/udev/udev.conf.

E' possibile forzare la creazione di un nodo in /dev/ creando il file in /etc/udev/devices in modo che venga copiato automaticamente da udev in /dev.

hotplug

Hotplug [http://linux-hotplug.sourceforge.net/] è un programma che aggiunge a GNU/Linux il supporto per l'inserimento dinamico di periferiche. Tramite gli opportuni segnali dal kernel esegue un autoloading dei moduli opportuni e quindi permette alle applicazioni di sistema di "vedere" tali periferiche. Supporta le periferiche USB, PCI, Firewire e puo effettuare il download del firmware dei dispositivi quando necessario. A partire dalla versione 2.6 del kernel Linux,

hotplug ne è divenuto parte integrante. Da esperienze recenti trovo che questa scelta sia stata un po' azzardata infatti con delle periferiche USB recenti (ram-disk) ho avuto, dopo il tentativo di riconoscimento automatico da parte di hotplug, dei messaggi di errore dal kernel con il blocco delle porte USB. Il sistema non ha subito un crash generale (e questa è gia una buona cosa) ma rimane la possibilità di un comportamento diverso per una diversa periferica.

Considerazioni

Come ho avuto già modo di osservare le innovazioni portate da questo progetto sono rivolte ad aumentare l'usabilità di GNU/Linux in ambiente desktop e giovano ai nuovi utenti meno smaliziati. Sicuramente sono inutili in un server dove l'installazione di questi programmi può addirittura portare problemi.Per quanto siano sviluppati correttamente, probabilmente, questi servizi aggiunti contribuiscono a rallentare le prestazioni del PC e potrebbero introdurre nuovi punti di errore (keep it as simple as possible).

[reference]

1. Home page del progetto utopia
2. Hardware desktop layer home page
3. DBUS home page
4. UDEV home page
5. udev and devfs - The final word
6. Hotplug [http://linux-hotplug.sourceforge.net/]

di Rudi Giacomini Pilon