

# Integración de hardware en el escritorio

## Una mirada al usuario desde GNOME

Álvaro del Castillo

Lambdaux Software Services S.R.L.  
Universidad Rey Juan Carlos  
Centro de Apoyo Tecnológico  
C/ Tulipán sn 28933 Mostoles, Madrid-Spain  
acs@lambdaux.com

**Resumen** El presente trabajo tiene como objetivo mostrar el estado actual de la integración del hardware en los escritorios libres, haciendo un especial hincapié en la visión del usuario y en general, en el escritorio GNOME. Muchas de las tecnologías que aparecerán a lo largo de este trabajo son compartidas por todos los escritorio libres, por lo que están siendo normalizadas en Freedesktop.

### 1. Visión general del hardware para los usuarios

Sin lugar a dudas una de las labores más duras de los desarrolladores de software es el ser capaces de entender como los usuarios finales de sus desarrollos van a entender la herramienta que se pone en sus manos, algo que habitualmente se conoce como el modelo mental del usuario. Cuanto mejor entendamos este modelo más sencillas de utilizar serán nuestras herramientas ya el usuario podrá aplicar sus procesos habituales de razonamiento utilizando el modelo mental deducido para ir aprendiendo y memorizando el uso de la herramienta.

Si nos centramos en el hardware de los equipos, este modelo mental de los usuarios pasaría por considerar el hardware como algo que se configura y activa sólo, y del que el usuario en cuestión sólo ha de sacar partido con las herramientas que hagan uso de él. Por ejemplo, si pensamos en un escáner, el usuario cuando lo compra y llega a su lugar de trabajo e intenta utilizarlo, piensa que con enchufarlo de forma correcta a la alimentación y a su equipo, podrá comenzar a capturar documentos desde una herramienta y podrá incorporarlos a su biblioteca de objetos multimedia de trabajo. Cualquier acción que se salga de estos pasos rompe con el modelo mental del usuario, con su ritmo de trabajo, le desvía de su objetivo principal, transferir material impreso a su equipo y, en definitiva, le hace alejarse de la experiencia satisfactoria que todo desarrollador de software siempre busca para sus usuarios.

En el mundo del software libre, y en concreto en el mundo basado en GNU/Linux, esta visión objetivo en muchas ocasiones no se alcanza. Los motivos son muchos, como podremos ir viendo a lo largo de este documento, desde que los fabricantes de hardware no facilitan los controladores para Linux para el

hardware, hasta que los sistemas de gestión de hardware en Linux terminan obligando a un conocimiento a bajo nivel del sistema, lo que imposibilita para la inmensa mayoría de las personas el uso de dicho hardware en Linux y, como consecuencia, el uso de Linux.

Una de las primeras descripciones del problema que existe en GNU/Linux con la gestión del hardware la llevó a cabo Havoc Pennington en su documento *Making Hardware Just Work* en el que se muestran varios casos de uso similares al descrito anteriormente con los escáneres y se propone una solución que ya se está implementado. A lo largo del documento vamos a ir mostrando todas las tecnologías que están permitiendo que cada vez estemos más cerca de un escenario en el que el usuario final del equipo, una herramienta, sea capaz de tener una experiencia adecuada.

En cualquier caso, tenemos que resaltar la complejidad del mundo del hardware, los miles de dispositivos que existen con todo tipo de características, y la velocidad de renovación del mercado del hardware.

## **2. Lograr que el hardware funcione en Linux para los ojos del usuario**

Actualmente, marzo de 2005, cada vez es más habitual que el hardware que existe en el mercado de la informática tenga algún tipo de soporte para Linux. Existen cientos de controladores en el núcleo de Linux, la pieza de software que cuya principal responsabilidad es gestionar el hardware, y muchos fabricantes de hardware publican los suyos propios. Al aumentar la cuota de usuarios, y en especial la cuota de usuarios de escritorios, los fabricantes de hardware cada vez tienen más presión en que su hardware tenga soporte en Linux. Pero, ¿es suficiente con el el hardware esté soportado? ¿Es esto lo que necesitan los usuarios?

Lo primero que debe de saber el usuario es si su hardware está soportado realmente por el núcleo de Linux y/o los diferentes proyectos que dan soporte al hardware, como por ejemplo CUPS o SANE. Para ello, el proyecto CompatibleLinux ha recopilado la mayoría de la información existentes de decenas de fuentes sobre el hardware con el objetivo de informar al usuario si es seguro o no comprar determinado hardware para usarlo en Linux. Además de saber si el hardware funciona o no, se dan indicaciones sobre el controlador y distribuciones que soportan dicho hardware, facilitando un poco más la labor del usuario. En la actualidad ya hay más de 16.000 dispositivos registrados, y se actualiza de forma frecuente. Cada distribución de Linux por su parte suele tener también una base de datos específica del hardware que soporta.

Los usuarios avanzados de Linux, una gran proporción aún, se han acostumbrado al uso de terminales desde los que realizar tareas como la carga manual de los controladores del hardware, o la modificación de ficheros de configuración para que los controladores se carguen en el inicio de la máquina. Pero un usuario final, ¿sabe lo que es un controlador?, ¿sabe qué controlador es el adecuado para un dispositivo de hardware?, es más, ¿debemos suponer que los usuarios deben

de tener estos conocimientos? Es evidente que si queremos que GNU/Linux se generalice y llegue a la sociedad en general, necesitamos eliminar todo este tipo de intervenciones por parte del usuario.

El objetivo es pues llegar a tener un entorno donde la gestión del hardware se realice de forma transparente para el usuario. Para lograr este objetivo, son muchas las piezas que deben de trabajar en conjunción, desde el núcleo Linux hasta la sesión de escritorio en la que el usuario trabaja. Vamos a ir analizando toda esta cadena para ver las soluciones actuales, su evolución y que experiencia podemos ofrecer hoy a nuestros usuarios.

### **3. El núcleo Linux y sus ayudantes**

Como ya hemos comentado, el núcleo Linux es la pieza clave en la gestión del hardware. Cuando la máquina inicia es el núcleo lo primero que se carga, una vez que se ha pasado la etapa BIOS del arranque, y es él el que va descubriendo el hardware pasando por su detección, localización de controladores y configuración inicial. Esta primera etapa es imprescindible en el proceso de terminar ofreciendo al usuario las funcionalidades de los dispositivos. Analicemos cada una de estas labores y su evolución en la versión 2.6 del núcleo.

#### **3.1. Detección del hardware**

Una vez que el núcleo se ha cargado en memoria comienza a analizar el sistema sobre el que está trabajando. Realiza unas primeras labores de inicialización de los sistemas básicos (buses, memoria, CPU ...) y va inicializando aquellos dispositivos para los que el controlador está directamente en el núcleo.

Una vez completada esta primera fase, se llega a tener un sistema con una funcionalidad suficiente para labores básicas, pero alejada de la que puede esperar un usuario final. Por ejemplo, si el controlador de la tarjeta de red del usuario no se haya incluido en el núcleo, el usuario no tendrá conectividad a su red, teniendo un equipo en muchos casos inútil. Y la tendencia a excluir cada vez más del núcleo controladores y su inclusión como módulos es cada vez mayor, con el objetivo de no malgastar la memoria del usuario y tener núcleos menos óptimos al incluir soporte para hardware que el usuario no tiene.

De los anterior puntos llegamos a la conclusión de que la detección del hardware no implica su puesta en funcionamiento, a menos que se añada a esta detección del hardware, la detección del módulo que lo hace funcionar, y su carga.

#### **3.2. Configuración del hardware**

Tradicionalmente han existido bases de datos de hardware que dada una información sobre un dispositivo, indicaban el controlador que lo hacía funcionar. Ejemplos son kudzu, discover o ldetect. Estos sistemas se basan en tener un motor de detección de hardware que unido a la base de datos de soporte de

hardware, cargan los controladores necesarios para hacer que el hardware funcione. El gran problema con estos sistemas es que cada uno tiene su base de datos independiente y, que los módulos que indican que han de ser cargados, pueden no estar disponibles en el sistema del usuario. Iniciativas como CompatibleLinux.org buscan unificar todas estas bases de datos, aunque el concepto de motor de detección de hardware asociado a estas herramientas se va quedando poco a poco obsoleto frente al nuevo motor de detección de hardware, el propio núcleo Linux y los nuevos mecanismos de hotplug.

Al igual que existen bases de datos de hardware para cada una de estas herramientas, el núcleo Linux tiene su propia base de datos, formada por la información de todos los dispositivos para los que tiene un controlador que los hace funcionar. Esta información es incluida dentro de los controladores por los propios desarrolladores del controlador, y cuando el controlador (módulo) es instalado en el núcleo y se resuelven sus dependencias (`depmod -a`), todos los dispositivos que funcionan con ese controlador pasan a ser incluidos dentro de la base de datos del núcleo. Esta base de datos es la que en realidad es totalmente funcional para el sistema del usuario, ya que se refiere al núcleo Linux que está utilizando el usuario. Pero, ¿quién utiliza esta base de datos?

### 3.3. Hotplug o el mecanismo de avisos de hardware del núcleo

Con la llegada del núcleo 2.6 han entrado varias piezas fundamentales para el panorama actual de soporte hardware que estamos desarrollando. Una de estas piezas claves es el sistema de avisos de hardware del núcleo: mediante un mecanismo de eventos, el núcleo va informado de todo el hardware que ha localizado en la máquina. Recogiendo estos eventos, y utilizando una base de datos que nos diga que módulo está asociado a qué dispositivo, deberíamos de ser capaces de cargar sin intervención del usuario todos los módulos que hacen funcionar el hardware.

Unido a este sistema de eventos, se encuentra el nuevo sistema de ficheros “sysfs” que muestra como una estructura de directorios bajo diferentes clasificaciones, el hardware que se encuentra en el sistema. Esta información será clave para que las capas superiores de software puedan localizar de forma sencilla el hardware.

El sistema actual que consume los eventos de hotplug que genera el núcleo cuando se añaden y quitan dispositivos (`add/remove`) es el programa “`/sbin/hotplug`”. Este programa tan sólo se encarga de entregar los datos del evento a los agentes adecuado. Estos agentes son actualmente programas en “shell” que leen la información del dispositivo que les ha dado el núcleo e intentan cargar el controlador adecuado para hacerlo funcionar, así como realizar otras acciones que pudieran ser necesarias (en los escáneres se modifican los permisos del dispositivo, por ejemplo). El hecho de que sean “scripts” muestra lo sencillo de la labor que hacen. En los apéndices de este documento se encuentran ejemplos de código que muestran como se implementan.

Finalizado el proceso de ejecución de los agentes de hotplug el hardware se habrá dejado ya lo más configurado posible para que se puede acceder a él desde

las aplicaciones. Pero para poder acceder a él, necesitamos encontrarle (habitualmente localizar el nombre de dispositivo) y, por otro lado, cuando se añade o elimina hardware del sistema, será necesario utilizar algún tipo de aplicación para hacer uso de él. Por lo tanto, aún nos faltan dos procesos, el de localización del punto de comunicación con el hardware y el de aviso al sistema de que hay nuevo hardware disponible en el sistema.

### 3.4. udev: sistema de nombrado persistente del hardware

Cuando una aplicación quiere acceder al hardware lo hace normalmente utilizando unos ficheros especiales que representan los dispositivos hardware. Estos ficheros se encuentran dentro del directorio “/dev” del sistema de ficheros. Tradicionalmente se incluían en este directorio los cientos de posibles dispositivos que se pudieran encontrar en un sistema genérico, así como un script “MAKEDEV” que permitía crear aún más. Este esquema poco escalable se ha ido modificando hasta llegar actualmente a un sistema dinámico, en el que los ficheros de dispositivos en “/dev” se crean bajo demanda cuando el dispositivo se encuentra en el sistema.

Para la creación de estos ficheros de dispositivos en “/dev” es vital que el núcleo informe de forma correcta de todo el hardware encontrado y que haya una aplicación que se encargue de crear los ficheros. Con la llegada del núcleo 2.6 y la unificación del modelo de dispositivos hardware bajo kobject, ha sido posible esta forma de funcionar y el nacimiento del sistema “udev”, cuya principal tarea es la de crear los dispositivos adecuados bajo “/dev” para que el software de capas superiores se encuentre ya creados los dispositivos y pueda hacer uso de ellos.

Es más, udev introduce una característica muy interesante. Es habitual que según a que puerto de la máquina enchufes un dispositivo usb, este se accede como un fichero de dispositivo diferente. Gracias a udev, se puede hacer que un dispositivo (se puede reconocer por los identificadores USB del dispositivo por ejemplo), se accede siempre de la misma forma, algo que simplifica enormemente la labor de las capas superiores de software, que pueden utilizar siempre el mismo nombre de dispositivo. O por ejemplo, se puede dar un nombre al dispositivo que permita el usuario identificarlo de forma sencilla y ayudar así a la aplicación a localizarlo.

Como ya hemos visto anteriormente, dentro de los scripts que se ejecutan siempre ante la llegada de nuevo hardware con hotplug, estaba el programa “udev.hotplug -> /sbin/udevsend” que se encarga de la creación de los ficheros de dispositivos adecuados en función de la configuración de “udev” y de una serie de reglas que se especifican dentro de “/etc/udev/”. El resultado final pues es que tenemos un sistema de nombrado para los dispositivos que facilita su localización.

Destacar que se está planteado el sustituir el script de “/sbin/hotplug” directamente por “udevsend” y que este se encargue de la creación de los dispositivos y de la ejecución de los agentes de hotplug.

### **3.5. HAL: la capa de abstracción de hardware**

Para facilitar la vida de los desarrolladores de aplicaciones que hacen uso de hardware, se ha creado la especificación HAL. El objetivo es ofrecer una API unificada de programación a los desarrolladores de aplicaciones de forma que puedan manejar de forma sencilla el hardware.

Con esta capa nos acercamos ya mucho a lo que sería el escritorio libre, el conjunto de aplicaciones que ofrecen al usuario una interfaz gráfica integrada para utilizar el sistema. El objetivo de HAL es que dichas aplicaciones usen la API que se proporciona desde HAL para acceder al hardware, de forma que HAL se puede encargar de todos los detalles de bajo nivel del sistema y permitir al programador centrarse en ofrecer al usuario las funcionalidades que necesita, y no en tener que realizar procedimientos complejos para poder soportar diferentes configuraciones de hardware.

Las labores principales en las que se centra HAL es en las de permitir a las aplicaciones descubrir y configurar el hardware que se encuentra en la máquina. Por ejemplo, una aplicación de gestión de fotos podrá preguntar a HAL como hablar con la cámara de fotos, o una aplicación de videoconferencia, cuál es el dispositivo de vídeo y el de audio.

Para que todo esto sea posible, HAL se compone de un servicio (demonio) con toda la información del hardware de la máquina, la gestión de su ciclo de vida de funcionamiento y la detección y monitorización de los diferentes buses del sistema para poder informar a las aplicaciones interesadas de cambios en el hardware. HAL, al igual que los agentes de hotplug, pueden realizar acciones dentro del sistema para poder configurar los dispositivos.

Como toda aplicación que lo único que quiere hacer es poner a disposición de las aplicaciones las funcionalidades del hardware, no intenta imponer ningún tipo de política en el uso de los dispositivos.

La información sobre el hardware que llega a HAL lo hace de hotplug y de udev. Con estas dos fuentes de información, unida a la información de su base de datos de información de dispositivos HAL es capaz de cumplir su gran objetivo, proporcionar toda la información interesante sobre el hardware a las aplicaciones del sistema.

### **3.6. DBUS: Comunicaciones entre procesos**

Para cerrar el conjunto de tecnologías que dan soporte a las tecnologías que nos permiten una gestión completa del hardware en Linux, vamos a describir DBUS.

DBus nos permite comunicar aplicaciones dentro de una misma máquina de una forma desacoplada y síncrona/asíncrona. DBus se base en crear una serie de canales de comunicación (buses) que permiten comunicar al sistema con las diferentes sesiones de usuario que pudiera existir, y a las aplicaciones de una sesión de usuario entre sí. Lo habitual es tener en un sistema un canal de sistema donde se producen las comunicaciones a nivel de sistema entre los procesos ejecutándose en la máquina, y hay otro canal específico de la sesión

de escritorio de usuario donde sólo se pueden comunicar las aplicaciones que se están ejecutando dentro de dicha sesión. Para la gestión de hardware, HAL utiliza el bus del sistema para comunicar a todos los procesos interesados en la máquina los eventos de hardware que se vayan produciendo.

Mostramos a continuación un pequeño ejemplo en Python de como se obtienen todos los dispositivos que se encuentran disponibles en el sistema, ejemplo proporcionado dentro de la especificación de HAL:

```
#!/usr/bin/python

import gtk
import dbus

def device_added(interface, signal_name,
                 service, path, message):
    [udi] = message.get_args_list ()
    print 'Device %s was added'%udi

def device_removed(interface, signal_name,
                  service, path, message):
    [udi] = message.get_args_list ()
    print 'Device %s was removed'%udi

bus = dbus.Bus (dbus.Bus.TYPE_SYSTEM)
hal_service = bus.get_service
                ('org.freedesktop.Hal')
hal_manager = hal_service.get_object
                ('/org/freedesktop/Hal/Manager',
                'org.freedesktop.Hal.Manager')

devices = hal_manager.GetAllDevices ()
for d in devices:
    print 'Found device %s'%d

bus.add_signal_receiver (device_added,
                        'DeviceAdded',
                        'org.freedesktop.Hal.Manager',
                        'org.freedesktop.Hal',
                        '/org/freedesktop/Hal/Manager')
bus.add_signal_receiver (device_removed,
                        'DeviceRemoved',
                        'org.freedesktop.Hal.Manager',
                        'org.freedesktop.Hal',
                        '/org/freedesktop/Hal/Manager')

gtk.main()
```

Vemos como el programa comienza por establecer una comunicación con el bus de sistema que crea de forma automática DBUS durante el inicio del sistema, y dentro de dicho bus localiza el servicio “org.freedesktop.Hal”. Una vez localizado el servicio, podemos comenzar a invocar las funciones del API DBUS que nos ofrece HAL. En este caso lo que hacemos es obtener la referencia al objeto que representa al dispositivo “Manager” que nos da acceso a todo el hardware de la máquina. Su API es algo más reducida que la del objeto “Device” que representa a un dispositivo hardware de la máquina. La API completa de HAL sobre DBUS en su versión 0.5 (en desarrollo y aún sólo en CVS) se encuentra en las referencias.

Además de mostrar todos los dispositivos que se encuentran dentro del sistema, este sencillo ejemplo se engancha a las señales de HAL que indican la aparición de nuevo hardware o su eliminación, de forma que cuando por ejemplo enchufamos y desenchufamos un ratón USB, recibimos los eventos de hotplug en nuestra aplicación, pero transformados a objetos HAL. En concreto, los eventos que recibimos son:

```
Device /org/freedesktop/Hal/devices/  
usb_usb_device_458_36_110_-1_noserial_0 was removed  
Device /org/freedesktop/Hal/devices/  
usb_device_458_36_110_-1_noserial was removed  
Device /org/freedesktop/Hal/devices/  
usb_device_458_36_110_-1_noserial was added  
Device /org/freedesktop/Hal/devices/  
usb_usb_device_458_36_110_-1_noserial_0 was added
```

Junto a los eventos, viene el UDI (identificador único) del dispositivo con el que podremos obtener el objeto del dispositivo. Por ejemplo, podemos preguntar a HAL todas las propiedades de un dispositivo que se acaba de añadir con una pequeña modificación en la gestión del evento de nuevo dispositivo añadido “device\_added”:

```
def device_added(interface, signal_name, service, path, message):  
    [udi] = message.get_args_list ()  
    hal_device = hal_service.get_object(udi,  
                                        "org.freedesktop.Hal.Device")  
    properties = hal_device.GetAllProperties();  
    for p in properties:  
        dev = hal_device.GetProperty(p)  
        print 'Found property %s: %s'%(p, dev)  
    print 'Device %s was added', udi
```

De estos ejemplos cabe resaltar que la API de DBUS puede ser accedida igualmente desde C que desde Python que desde cualquier lenguaje que tenga enlaces a DBUS. La API DBUS queda un poco escondida dentro de la unión de `hal` a `dbus`, aunque está bien especificada dentro de la especificación de HAL.

La aplicación “hal-device-manager” es un excelente ejemplo de visualizador de hardware, demasiado complejo para un usuario final, pero adecuado para usuarios experimentados que quieran conocer al detalle el hardware que se encuentra en la máquina.

#### 4. El uso de HAL dentro del escritorio GNOME

Hasta el momento hemos visto el conjunto de tecnologías que nos permite a los desarrolladores de aplicaciones una gestión del hardware de forma mucho más sencilla y más completa que la que antes existía, donde en cada aplicación había que introducir código específico de gestión de hardware. Pero, ¿se están ya utilizando estas tecnologías? ¿hay ejemplos de su uso?

El mejor ejemplo de uso de estas tecnologías es “gnome-volume-manager” (gvm), cuya principal labor es permitir que dentro de GNOME, la gestión de volúmenes en caliente se realice de forma amigable. Un volumen es por ejemplo una partición dentro de un llavero USB, o los contenidos de una cámara digital, o un CDROM. Cuando un nuevo volumen se incorpora al sistema, por ejemplo al pinchar un llavero USB, se produce un evento de hotplug, que termina llegando tanto a HAL como a udev.

Originalmente lo que se hacía era que HAL modificaba el fichero “/etc/fstab” y añadía una nueva entrada para el nuevo dispositivo detectado, activando como opción que los usuarios la pudieran montar. Tras ello, gvm se encarga de montar el nuevo volumen y realizar la acción especificada para él en las propiedades de gvm. El nuevo volumen lo monta donde HAL le indique mediante la propiedad “volume.mount\_point”. En este caso del llavero USB este valor es “/media/usbdisk”. HAL realiza la labor de modificar “/etc/fstab” utilizando el ayudante “fstab-sync” que se encarga de añadir en “/etc/fstab” las entradas necesarias para montar los volúmenes nuevos que HAL vaya detectado. gvm simplemente tendrá que invocar a “mount” con el dispositivo ya que en “/etc/fstab” estará especificado el punto de montaje del dispositivo.

En los últimos meses ha aparecido una nueva aplicación llamada “pmount” que es capaz de montar volúmenes sin necesidad de que estos se encuentren en “/etc/fstab”. Además, incluye soporte de alguna política, como la de bloqueo de acceso a los CDROMs cuando están grabando. Esta herramienta ya es la que se usa en Debian dentro de gvm, que ha sido modificado para utilizar “pmount-hal” al que basta con indicarle el UID del volumen para que realice toda la labor de montar el dispositivo.

Una vez que se tiene acceso al dispositivo gvm puede realizar diferentes acciones en función del tipo de contenidos que tenga: mostrar los contenidos, reproducir los vídeos o fotografías que incluya ... También es posible lanzar el grabador de CDs o el reproductor de música si lo que hay son canciones. En definitiva, gracias a gvm, el usuario se ve aliviado de las tareas de hacer que los contenidos estén disponibles en el sistema, y sólo tiene que utilizarlos.

## 5. No sólo de volúmenes vive el escritorio

Es probable que llegados a este punto, el lector haya generalizado lo que ocurre con los eventos que permiten gestionar los volúmenes a otros tipos de eventos. Con la llegada del núcleo 2.6.10 se incluyó un mecanismo para informar de eventos que se producen en el núcleo y queremos que se informe de ello a las aplicaciones en espacio de usuario interesadas. Lo cuenta Robert Love, desarrollador del núcleo, de g-v-m y autor de este mecanismo, en su blog el 1 de Noviembre de 2004.

Con este nuevo sistema de eventos, el núcleo puede informar a las aplicaciones de eventos que las pueden afectar, como por ejemplo, que se ha localizado una conexión de red (y hace falta configurarla), que la temperatura de la CPU es muy alta (y hay que bajar la velocidad de la CPU o encender los ventiladores) ... o cualquier otro evento que seguro que le puede ocurrir al lector. Se tiene en mente ir desarrollando en GNOME un capa que se encargue de recibir estos eventos y que se los puedan comunicar a las aplicaciones directamente. Aún es pronto para ver como será finalmente este sistema, si se usa dentro del núcleo por parte de los desarrolladores de controladores y si finalmente las aplicaciones escucharán los eventos, pero al menos las grandes piezas ya están desplegadas.

## 6. Pasado, presente y futuro

GNU/Linux viene de un pasado fuertemente técnico. Unix nunca ha sido un sistema operativo para usuarios sin conocimientos informáticos, y es una cultura que se refleja a lo largo y ancho de todo el sistema. Sin embargo, la popularización del software libre ha llevado a que los sistemas Unix lleguen a todo tipo de usuarios, y esto ha provocado un nuevo reto: ser capaces de hacer de GNU/Linux un sistema sencillo de utilizar. El trabajo a realizar es muy grande, pero las bases se van sentando y dentro del mundo hardware, hemos pasado de un modelo en el que el usuario se tenía prácticamente que compilar el núcleo a medida de su hardware, al uso generalizado de núcleos comunes con soporte en módulos para cientos de dispositivos. Los usuarios por lo general ya usan el núcleo que viene con su distribución y cada vez es más raro encontrar hardware que no funcione en Linux. Proyectos como [Compatiblelinux.org](http://Compatiblelinux.org) están ayudando aún más a mostrar que el hardware que ya funciona sin problemas en Linux.

Actualmente la mayoría del hardware funciona, pero para hacerlo funcionar, aún se necesitan en muchos casos de realizar labores fuera del alcance de la gran mayoría de usuarios finales. Las tecnologías que hemos mostrado a lo largo del documento muestran un camino claro a seguir, y a día de hoy, ya han facilitado labores haciendo posible que usuarios puedan ver fotografías en Linux, puedan grabar CDs o puedan usar sus llaveros USB de forma sencilla.

Del futuro podemos esperar que cada vez se vayan enriqueciendo más las políticas de uso de los dispositivos dentro de HAL así como la información que se dispone sobre el hardware. Según HAL se vaya estabilizando, nuevas aplicaciones pasarán a utilizarlo y esta unificación provocará sinergias que ayudarán a acelerar

el ritmo de adopción de HAL como mecanismo de comunicación con el hardware. Por ejemplo, aplicaciones como el gestor de impresión, deberá de hablar con HAL para saber que impresoras se han detectado y, recibirá notificaciones de HAL como que la impresora se ha quedado sin papel.

Sin lugar a dudas son aún udev, hotplug y HAL tecnologías que están madurando, con el núcleo de Linux, y que muestran un camino claro a seguir pero que aún no está listo en general para depender en las versiones estables de estas tecnologías. Como muestra, en 2.6.11 se han realizado actualizaciones importantes en la forma de trabajar de hotplug que sin duda, creará algún quebradero de cabeza en las migraciones de los sistemas actuales en producción. Para acabar, felicitar a GNOME 2.8 por ser el primer escritorio en comenzar a hacer un uso serio de estas tecnologías y a Ubuntu Warty por ser la primera distribución donde se integraron estas tecnologías y que pude disfrutar.

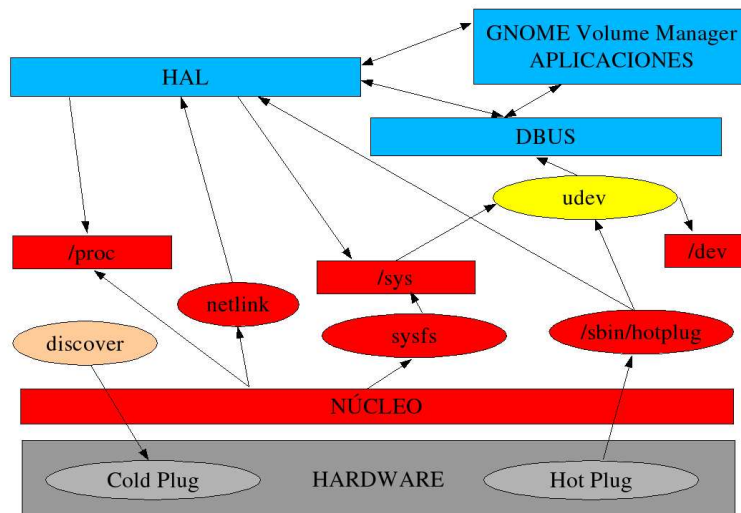


Figura 1. Arquitectura de gestión de hardware en Linux

## A. El programa /sbin/hotplug en Debian Sarge

El programa “/sbin/hotplug” es invocado por el núcleo cuando se produce la adición o eliminación de elementos de hardware (kobjects). Se pasan una serie de parámetros a este programa (también se pasan datos como variables de entorno). El primer parámetro especifica el tipo de dispositivo y se utiliza para ejecutar los agentes específicos de ese tipo de dispositivo. Por ejemplo, si pinchamos un ratón usb, se pasa como primer parámetro “usb” lo que provocará la ejecución

de los agentes hotplug USB. Siempre se ejecuta al final cualquier agente en el directorio de agentes genéricos (default).

```
DIR="/etc/hotplug.d"

for I in "${DIR}/$1/*.hotplug "${DIR}/default/*.hotplug ; do
    test -x "$I" && "$I" "$1"
done

exit 0
```

El lector interesado puede modificar “/sbin/hotplug” para que por ejemplo vuelque el entorno y los parámetros que se pasan al script. Un ejemplo sería:

```
DIR="/etc/hotplug.d"

echo "HOTPLUG $*" >> /tmp/hotplug
set >> /tmp/hotplug

for I in "${DIR}/$1/*.hotplug "${DIR}/default/*.hotplug ; do
    test -x "$I" && "$I" "$1"
done
```

Para el correcto funcionamiento de “udev” y “hal” se incluyen agentes genéricos de ambos sistemas que se invocan siempre que sucede algún evento de hotplug, de forma que pueden realizar las acciones precisas. Por ejemplo, hal informará a las aplicaciones interesadas en ese tipo de hardware. Y “udev” creará los ficheros de dispositivos necesarios para poder acceder al dispositivo.

```
acs@delito:/etc/hotplug.d/default$ ls -l
total 4
lrwxrwxrwx  1 root  root   24 2005-02-22 22:00 20-hal.hotplug ->
/usr/lib/hal/hal.hotplug
-rwxr-xr-x  1 root  root 2924 2004-09-28 02:36 default.hotplug
lrwxrwxrwx  1 root  root   14 2005-02-22 22:00 udev.hotplug ->
/sbin/udevsend
```

## B. Agentes de hotplug en Debian Sarge

Los agentes en Debian Sarge se encuentran incluidos dentro del directorio /etc/hotplug. Se llaman desde “/etc/hotplug.d/default”. Cuando por ejemplo llega un evento de la clase “usb” se llama al agente “/etc/hotplug/usb.agent” que se encarga de hacer todo lo posible para dejar el hardware funcionando. Por ejemplo, utiliza la base de datos de hardware del núcleo para localizar y cargar los módulos adecuados para el hardware que se ha añadido. Y no sólo esta base de datos, si no otras como por ejemplo la de escáneres o la de cámaras de fotos, que mantienen sane y libphoto2 respectivamente.

Un caso especialmente interesante es el de los escáneres. A partir del núcleo 2.6.4 se eliminó el controlador “scanner.ko” del núcleo y toda la funcionalidad se llevó a espacio de usuario, utilizando la librería libusb. Pero hay algunas acciones necesarias para que los usuarios puedan acceder al dispositivo del escáner, como habilitar permisos de acceso al dispositivo, que se hacen desde hotplug cuando se detecta que lo que se ha pinchado ha sido un escáner. Para ello, se utiliza la base de datos “libsane.usermap”. En el caso de estar el dispositivo especificado en esta base de datos, se ejecuta el script “libusbscanner” que se encarga de esta gestión de permisos. Incluimos como ejemplo este pequeño “script” a continuación.

```
# Arguments :
# -----
# ACTION=[add|remove]
# DEVICE=/proc/bus/usb/BBB/DDD
# TYPE=usb

# latest hotplug doesn't set DEVICE on 2.6.x kernels
if [ -z "$DEVICE" ] ; then
  IF='echo $DEVPATH |
      sed 's/^(bus\/usb\/devices\/\)\(.*\)-\(.*\)\/2/'
  DEV='echo $DEVPATH |
      sed 's/^(bus\/usb\/devices\/\)\(.*\)-\(.*\)\/3/'
  DEV='expr $DEV + 1'
  DEVICE='printf '/proc/bus/usb/%.03d/%.03d' $IF $DEV'
fi

if [ "$ACTION" = "add" -a "$TYPE" = "usb" ]; then
  chown root:scanner "$DEVICE"
  chmod 0660 "$DEVICE"
fi

# That's an insecure but simple alternative
# Everyone has access to the scanner

# if [ "$ACTION" = "add" -a "$TYPE" = "usb" ]; then
#   chmod 0666 "$DEVICE"
# fi
```

## Referencias

1. Cambios en /sbin/hotplug: <http://lwn.net/Articles/123932>
2. Nombres persistentes con udev: <http://www.linuxjournal.com/article/7316>
3. Especificación de HAL 0.5 CVS: [http://cvs.freedesktop.org/\\*checkout\\*/hal/hal/doc/spec/hal-spec.html?only\\_with\\_tag=hal-0\\_4-stable-branch](http://cvs.freedesktop.org/*checkout*/hal/hal/doc/spec/hal-spec.html?only_with_tag=hal-0_4-stable-branch)

4. API DBUS de HAL: [http://cvs.freedesktop.org/\\*checkout\\*/hal/hal/doc/spec/hal-spec.html?only\\_with\\_tag=HEAD#dbus-api](http://cvs.freedesktop.org/*checkout*/hal/hal/doc/spec/hal-spec.html?only_with_tag=HEAD#dbus-api)
5. Capa de Eventos del Núcleo: <http://rlove.org/log/2004110101.html>