REMODELANDO /etc

Carlos Manzanedo Rueda y Jordi Polo Carrés
Revisión: Pedro-Ángel González Rueda

23 de septiembre de 2003

Una nueva visión del directorio /etc, la configuración de los sistemas unix y los sistemas distribuidos.

INTRODUCCIÓN.

Con la fuerte expansión en todos los campos que está sufriendo GNU/Linux, el ámbito donde encontrar una máquina GNU/Linux más corriente ha pasado de ser un sistema aislado a agrupaciones de éstas (granjas de servidores, oficinas y aulas), pero no existen muchas herramientas que permitan gestionar de manera sencilla esta cantidad de máquinas. También estamos faltos de herramientas que nos permitan manejar nuestros ficheros de configuración de una manera cómoda. Creemos que éste es uno de los grandes problemas a los que nos enfrentamos ahora mismo y que estas herramientas se necesitan con urgencia.

La situación actual es que hace falta un experto que conozca muchos pequeños formatos de ficheros y que realize una gran cantidad de trabajo rutinario.

Esta ponencia pretende mostrar algunas de las ideas que están surgiendo de un nuevo modelo de trabajo en la utilización y diseño de los ficheros de configuración de forma que las aplicaciones accedan a ellos con sencillez.

Para demostrar la implementación práctica, hemos usado estas ideas para realizar una distribución basada en Debian y de codigo GPL, así como las aplicaciones creadas para dicha distribución.

No pretendemos imponer un nuevo estándar, sino exponer nuevas posibilidades a los desarrolladores y proponer ideas para situaciones recurrentes como acceso a ficheros de configuración por parte de las aplicaciones y el diseño de estos ficheros.

Con el modelo tratamos de dar un paso más con los sistemas GNU/Linux hacia los entornos distribuidos, donde un sistema no está compuesto por los servicios de una máquina, sino por los de muchas máquinas que se gestionan y comportan como un sólo sistema.

El encuentro con la situación.

A lo largo de nuestra experiencia en la administración, con cualquier tipo de sistema hemos encontrado siempre la misma situación; la falta de herramientas escalables y mantenibles para poder actuar sobre muchas máquinas sin tener que ir una a una configurando los mismos cambios o controlando su estado.

Cuando hemos necesitado administrar una red de *routers*, una granja de máquinas, los servidores corporativos o departamentales de una empresa o las estaciones de trabajo de un departamento de diseño, por poner algunos ejemplos; las posibilidades van desde hacer un conjunto de *scripts* ad hoc hasta adoptar y modificar alguna herramienta existente.

Los *scripts* nos permitían administrar varias máquinas simultáneammente, pero carecían o de generalidad o de funcionalidad, y siempre son poco escalables y mantenibles.

Las herramientas de configuración, aunque en su mayoría con un funcionamiento impecable, nos presentaban también algunos inconvenientes: en general reescribían los ficheros de configuración, de modo que la posterior edición manual se hacía farragosa. También seguía siendo necesario ir máquina a máquina y el mantenimiento de la aplicación era muy complejo debido a la heterogeneidad de los sistemas y de las aplicaciones.

Es una situación que claramente empeora cuando se tiene más heterogeneidad y cantidad de máquinas o servicios. Si con una pequeña cantidad de máquinas, una configuración como la que proponemos es deseable, en la administración distribuída se vuelve prácticamente imprescindible.

De la situación a las necesidades.

Durante los tres años que nuestro grupo de trabajo ha estado estudiando la problemática de los sistemas distribuidos, hemos encontrado una y otra vez que la necesidad de administración pasa por la capacidad de:

1. Gestionar simultáneamente los datos y configuraciones de varias máquinas como si fueran una sola.

2. Crear sistemas de toma de decisiones distribuidas capaces de manejar los datos de todo el sistema y presentarlos a los administradores de una manera sencilla, así como tomar decisiones simples de manera autónoma.

3. Crear herramientas que permitan que todo esto sea transparente para el usuario y el administrador novel y personalizable a todos los niveles para cualquier desarrollador o administrador avanzado.

Como primer paso, es necesario que el trabajo sobre los datos sea abierto y sencillo. Además se necesita que las configuraciones sigan siendo en texto plano para poder editarlas a mano o con *scripts*. Después de considerar varios formatos y estilos de esquemas de datos, llegamos a la conclusión de que un modelado basado en XML es el que mejor encaja con nuestras necesidades, como explicaremos más adelante.

/etc, el directorio crítico.

En el directorio /etc se deben guardan todas las configuraciones del sistema y algunas otras cosas. En general, las aplicaciones UNIX hacen uso de ficheros de texto plano para guardar sus configuraciones. Cada fichero posee su propia sintaxis, sin seguir ningún tipo de uniformidad o estándar; esto hace que el trabajo sobre los datos que gestionan nuestros sistemas sean difíciles de manejar.

Un problema del directorio /etc es la falta de uniformidad en los ficheros de los que consta; cada aplicación e incluso cada distribución ubica sus propios ficheros de configuración en /etc sin seguir un estándar o una sintaxis básica. La confusión que esto genera hace que se coloquen comentarios autoexplicativos kilométricos en cada fichero, lo cual es de agradecer, pero es indicativo de que el caos es tácito y aceptado. Por tanto no sólo cada fichero tiene su propia sintaxis sino que de un sistema a otro el fichero puede existir o no o encontrarse en lugares distintos.

No existe una norma para acceder a /etc, no hay una forma evidente de saber dónde está lo que se está buscando y cómo modificarlo. A lo largo de la ponencia veremos otro punto de vista de cómo podría ser el directorio /etc o, al menos, parte de él. Presentaremos esquemas que hemos utilizado en algunas aplicaciones que dejarán constancia de la utilidad del modelo, para la gestion de red o el particionado de discos duros en nuestros sistemas.

De las necesidades a la solución práctica.

Debíamos comprobar que un modelo XML nos servía para todo lo que necesitábamos. Por tanto, lo pusimos en práctica, usamos el modelado para almacenar en /etc determinados datos de configuración del sistema, como el *hardware* de la máquina, la configuración de usuarios y la configuración de red. Luego utilizamos estos ficheros en una serie de aplicaciones para realizar parte de la administracion del sistema en una distribución que estábamos desarrollando y el resultado fue muy prometedor.

Gracias a ello existe ya un conjunto de herramientas suficiente como para asegurar que el trabajo con este modelo de desarrollo es viable y rápido. Explicaremos la base de funcionamiento de estas herramientas en unos instantes.

Más ventajas del modelo.

Cada programa que quiera configurar un apartado del sistema tiene que reescribir codigo, igual que hacen el resto de los programas que configuran esa misma parte del sistema, para acabar generando un fichero en el directorio /etc o simplemente crear un nuevo interfaz gráfico (UI, *user interface*).

Cada uno de nuestros sistemas está compuesto por tres partes. Por un lado una biblioteca o programa capaz de leer datos en XML y hacer los cambios en el sistema, o al revés, leer los datos del sistema y generar un XML. Luego están los propios datos modelados en ficheros XML. Por último los UIs que se encargan de interpretar los datos del modelo XML y presentarlos al usuario o escribir ficheros de configuración en XML para que se realicen cambios sobre el sistema que ha indicado el usuario. Este modelo de diseño es conocido como *model view controller* (MVC).

Separación de UI y sistema.

Esta división y el modelado de datos en XML permite que los programadores de UIs dediquen su tiempo sólo a hacer programas gráficos cada vez más intuitivos y atractivos, sin tener preocuparse de cómo utilizar los datos que van a configurar la máquina. Así sólo tendrán que recabar datos del usuario, escribirlos en un XML y llamar a una biblioteca o programa para que los procese.

Esto puede hacer que un programador no tenga que conocer tantos detalles, sino que se puede dedicar a la configuración o a la construcción de los programas sin tener que dominar los dos aspectos.

Bibliotecas que sepan como trabajar con los datos.

Por otro lado, los programadores del sistema pueden avanzar en su trabajo sin temer que todas las herramientas desarrolladas deban ser reescritas; basta con que mantengan compatibilidad en el modelo de datos XML.

Para ello deben existir una serie de bibliotecas y programas que se encarguen de trabajar con los datos modelados en XML, los interpreten y configuren las aplicaciones o el sistema tal y como exigen los ficheros de configuración.

Optimización de recursos.

GNU/Linux tiene infinidad de aplicaciones que configuran nuestro sistema, en modo texto y en modo gráfico; y muchas de ellas configuran las mismas cosas. El modelo mostrado hace que la escritura de los ficheros de configuración sea sencilla, de manera que cierta parte del código, la parte de las bibliotecas, sólo se tenga que tocar para hacer nuevos añadidos, pero nadie debería tener que reescribir la lógica del sistema para configurar un mismo servicio o aplicación y cuando sea necesaria la reescritura se pueda reutilizar gran parte del código.

DECLARACIÓN DE INTENCIONES.

Sistema de gestión sencillo.

La sintaxis de los ficheros de configuración es diferente entre aplicaciones, incluso entre las distribuciones, esto hace que el manejo de todos los datos de /etc tenga que ser casi artesanal.

¿Cuáles son nuestras intenciones, nuestra meta? Algo tan sencillo como un sistema que nos dé flexibilidad y simplicidad.

La gestión sobre unos pocos de estos ficheros es sencilla, ya que en general todos ellos están en texto plano y bien comentados. El problema llega cuando tratamos de gestionar muchos datos. Para configurar acciones de servicios conjuntos entre máquinas, el sistema se vuelve en nuestra contra y requiere del conocimiento de varias docenas de ficheros por parte del administrador. Si añadimos que el sistema puede estar compuesto por el centenar de máquinas del ejemplo, gestionarlo requiere el trabajo de varios administradores manipulando algunos miles de ficheros.

Para el programador es un problema aún mayor; no poder reutilizar código y tener que tratar cada esquema de datos como algo aislado hace que pierda más tiempo tratando los datos que efectuando la configuración del sistema con ellos; e incluso hace que tareas como la de desarrollar un programa que gestione, aunque sea básicamente, una máquina GNU/Linux sea algo implanteable o esté hecho a base de módulos que en la mayoría de los casos no tienen nada que ver entre sí. O simplemente han desperdiciado más tiempo del que debieran en el tratamiento de datos, restando tiempo a otras tareas. Y esta "pérdida de tiempo" en el modelado de datos se repite para cada programador que realice una aplicación semejante.

Para resolver los problemas de manera sencilla, comenzamos por desarrollar un par de herramientas que demostrasen si el esquema era completo. Estas herramientas han sido una distribucion, que hace uso de XML en toda la instalación, y las dos aplicaciones que pondremos por ejemplo.

Una de ellas es una muy típica: un gestor de red, que nos daría una idea de las ventajas que se pueden obtener de este esquema. La segunda aplicación es CoolDrive, un particionador de disco que haciendo uso del modelo puede ir más allá que "sólo" tratar las particiones de los dispositivos de almacenamiento.

Como el contenido de /etc son datos, ¿con qué diseñar el esquema? Ya hemos avanzado que para gestionar todos los datos de las aplicaciones de manera sencilla hemos utilizado XML.

Esto nos permitió manejar en todo momento desde UIs gráficos a *scripts* en el proceso de configuración de cualquiera de nuestras aplicaciones. Los *scripts* o programas escribían los datos en XML o leían los datos de un XML. Cuando querían actuar sobre el sistema llamaban al programa o biblioteca pertinente para que leyese esos XMLs y se realizasen las acciones.

La sencillez que daba separar las herramientas en GUI, datos y bibliotecas de manejo de datos, permitió actuar de manera independiente sobre cada uno de ellos.

Trataremos los puntos de este esquema a lo largo de la ponencia, para ver más en detalle qué ventajas aporta a cada una de las tres secciones.

Que el modelado de datos, sin duda la parte más crítica haga uso de XML hizo que pudiéramos gestionar todos los ficheros de nuestras aplicaciones de manera eficiente y sencilla, permitiendo realizar de forma trivial muchos de los procesos de automatización que ahora son tan díficiles de efectuar.

Por último, somos conscientes de que el sistema actual de /etc, tiene una gran *inercia*, y que cualquier cambio requeriría el tiempo y acuerdo de mucha gente, así que nuestra meta es mostrar los resultados de nuestra investigación para dar a conocer un modo viable y sencillo de solventar en líneas generales los problemas que hemos comentado.

Sobre cualquier plataforma (programación).

Editar a mano ficheros con formato de texto plano en GNU/Linux es imprescindible, pero también debería serlo el poder hacer uso de *scripts* para configurar aplicaciones, o programas de cualquier tipo.

Otra de las ventajas de este esquema es que al tener los datos en XML, tanto las bibliotecas como los interfaces de usuario sólo necesitarán poder leer o escribir XML lo cual está ya implementado en todos los lenguajes mayoritarios. Así que en el proceso de creación de, por ejemplo, un particionador, la biblioteca podría estar en C y el interfaz gráfico en Python, con bindings de QT o GTK. El caso es que la gestión tanto de bibliotecas como de UIs o *scripts* sea lo más flexible posible.

Transición.

Algunas dificultades que aparecerían en una solución como este proyecto sería empezar y que la gente lo utilizase. Realmente existen dos puntos importantes a tener en cuenta en la transición.

El primero es que ya existen muchos ficheros que están muy formateados. Por ejemplo, el caso de la configuración de las XFree es un claro caso de que costaría realmente poco poner unos cuantos tags en el XF86Config y ajustarlo al estándar XML (esta posibilidad se discutió en la propia lista de desarrollo de xfree). Así que no tendría dificultad hacer esa transicion en terminos de datos; pero no es así en todos los ficheros. El modelado de muchos datos es menos sencillo de lo que parece, aunque el resultado final tenga muchas ventajas.

Por ejemplo, imaginemos que existiera un único fichero de configuración de MTA en GNU/Linux para todos los servidores de correo. En este fichero existiría una parte común que compartirían todos los servidores y luego partes únicas por cada servidor (extensiones) para dar distintas funcionalidades. La manera de representar esa parte común presentaría un gran conflicto. Otro ejemplo; imaginemos que quisiéramos representar la información de *hardware* de la máquina en /proc en un fichero XML. Existirían muchas maneras de hacerlo, todas ellas con puntos fuertes y débiles.

El segundo punto, es la *inercia*. No sería posible que nadie cambie de esquema en sus /etc de la noche a la mañana. Lo que sí sería posible es que, durante la transición, al configurar un fichero clásico de /etc cambiase su homologo en XML y viceversa, de tal manera que ambos se mantuviesen sincronizados. Existen demonios como FAM (*File Alteration Monitor*) que permiten efectuar acciones cuando un fichero cambia, de manera que con el cambio de cada uno de los ficheros podríamos modificar el XML o el fichero de texto plano para que actuase en consecuencia.

Con inercia nos referimos a la gran cantidad de programas, *scripts* y bibliotecas que dejarían de funcionar correctamente al cambiar el formato de algún fichero fundamental, así como acostumbrarse a los nuevos ficheros por parte de los administradores de los sistemas.

Una mirada en el futuro

¿Cuáles son nuestras verdaderas intenciones? Algo tan sencillo como poder trabajar con los datos de una manera sencilla. En un principio, por todo lo enunciado arriba, pero a las verdaderas intenciones nos llevó un tren de pensamiento parecido a éste: imaginemos toda la configuración de un sistema hetereogéneo de ordenadores como una oficina o similar modelada en su mayoria en XML y con un sistema distribuido preparado para gestionar los datos y acciones:

- La configuración de todas las máquinas se podría guardar y reproducir en cualquiera de ellas de una manera muy sencilla.
- La configuración de las máquinas en un estado concreto y en un momento concreto se podría salvar para poder volver atrás con todo lo que ello implica. Naturalmente para poder realizar esto no solo se necesitan las configuraciones sino que harían falta más cosas, pero esta sería la base para hacerlo posible con facilidad.
- Podrían configurarse muchas máquinas aplicando cambios estándar sobre los XML, utilizando XLSTs y otras herramientas XML que haría el cambio sencillísimo.
- Podría, una vez detectada la caída de una máquina, utilizarse su configuración para levantar el servicio en otra de las máquinas del sistema.

- Un ejemplo muy útil para demostrar la potencia, flexibilidad y viabilidad de la solución expuesta es la siguiente situación:

  El administrador del sistema, o el propio sistema, determina que se necesita espacio de almacenamiento de un tipo determinado. El sistema o el administrador eligen el nodo que contendrá el almacenamiento; con un interfaz gráfico sencillo o un programa de toma de decisión automático se genera un XML de configuración estándar de almacenamiento que se pasará al nodo seleccionado para que se configure como sea necesario. Por otro lado genera un fichero XML para la configuración de los clientes de modo que puedan comenzar a trabajar de manera transparente con ese espacio de almacenamiento. Para todo esto, sólo habría sido requierido por parte del administrador una orden "necesito almacenamieto".

Es decir, que el motivo de poder manejar los datos de manera sencilla es el de poder crear un sistema de gestión flexible, sencillo y distribuido.

Poder hacer experimentos de simulación utilizando XML-RPC para particionar discos duros, administrar los usuarios de muchas máquinas a la vez, o crear sistemas similares a *corba* o *dcop* para la gestión de configuración del sistema. Todo ello sin perder de vista el punto de partida, que es un gran conjunto de ficheros que son editables por los administradores y que deberían seguir siéndolo.

La idea final y principal de este proyecto es que teniendo un acceso sencillo a los datos y un acceso distribuido sobre las máquinas se podría crear una capa de abstracción de complejidad variable para realizar la gestión del sistema de manera automática. Esta capa contendría la suficiente inteligencia para tomar decisiones sobre cómo debe ser administrado el sistema, cuándo necesita almacenamiento, cuándo tiene que guardar configuraciones, cómo deben administrarse los usuarios, o simplemente la calidad de servicio sobre la red que debe tener el sistema, como sistema en sí y nunca como un conjunto de máquinas sueltas, que es realmente el estado en el que estamos actualmente. De manera que nuestro esquema presenta los siguientes requisitos:

- Datos fáciles de manipular por administradores.
- Manejo de datos sencillo para los programadores de UIs.
- Manejo de datos sencillo a nivel de sistema distribuido.
- Una capa de abstracción que sepa cómo manejar los datos y qué representan.
- Adicionalmente, una capa capaz de tomar decisiónes acerca de qué y cómo debe actuarse con esos datos y avisar al administrador y al resto del sistema de las acciones a seguir o que efectúe dichas acciones.

MODELADO XML.

¿Por qué XML?

El porqué de la eleccion de XML se explica fácilmente exponiendo algunos de los escollos con los que se encuentra este modelo de desarrollo:

- Tiene que ser flexible, sencillo y comprensible (principio Kiss aplicado a grandes desarrollos).
- Se tiene que poder editar y ver desde editores éstandar para continuar con la administración actual.
- Tiene que ser escalable. Si se añaden funcionalidades a una aplicación, el modelado debe poder escalar sin muchos problemas.
- El manejo de los ficheros debe ser sencillo para que sea posible trabajar sobre muchas máquinas.
- Tiene que ser un estándar abierto.

XML resuelve todos esos problemas y unos cuantos más. Existen otras alternativas, pero el sentido en el que se está moviendo la representación de datos hace de XML la mejor opción actualmente. Sobre todo necesitamos flexibilidad. Realmente no conocemos la dificultad total que puede crearse con el cambio de parte de /etc a este esquema, ni cómo de difícil podría ser de mantener. Así que forzosamente debe ser flexible, escalable y estándar.

Ventajas que nos proporciona el uso de XML.

- Estándar.
- Escalable y flexible.
- Editable como texto.
- Soporta unicode.
- Manejable con *scripts*..
- Cualquier lenguaje de programación mayoritario tiene bibliotecas para acceso a XML.
- Se puede guardar y trabajar con XML comprimido o en bases de datos.
- XML-RPC como sistema de llamadas a procedimiento remoto puede ser utilizado de forma muy sencilla para dar el paso a sistemas distribuidos.
- Existen muchas herramientas para gestionar XMLs que ya se pueden usar.
- Esquemas para validar datos (DTD).
- XLST para modificar grandes cantidades de código XML de manera sencilla.
- Otras ventajas del estándar XML como son los Xpath o Xlink
- La representación de datos para interfaces web es casi inmediata.

Un primer ejemlo de XML que utilizamos en la distribucion es el siguiente. Durante el arranque de la distribución y en el futuro para cada arranque de la máquina, se obtiene a través de una biblioteca que desarrollamos sobre *libdiscover* el siguiente esquema XML que representa parte del *hardware* de la máquina (el esquema mostrado es parcial).

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AUTOMATICALLY_GENERATED_XML>
    <hardware>
        <cpus>
            <cpu>
              <id>1</id>
              <vendor>AuthenticAMD</vendor>
              <model>AMD Athlon(tm) MP 1800+</model>
              <bogomips>3047.42</bogomips>
              <bugs>(none)</bugs>
              <frequency>1526</frequency>
              <cache> 256 KB</cache>
              <options>[HAS_FPU:HAS_3DNOW:HAS_MMX]</options>
            </cpu>
        </cpus>
        <memory>
            <memory>513020</memory>
            <swap>996020</swap>
        </memory>
        <cdroms>
            <cdrom>
              <vendor>Unknown</vendor>
              <model>CD-224E</model>
              <device>/dev/hda</device>
              <module>ignore</module>
              <bus>ATAPI/IDE</bus>
            </cdrom>
        </cdroms>
        <videocards>
            <videocard>

              <vendor>ATI Technologies, Inc.</vendor>
              <model>3D Rage XL [264XL GR]</model>
              <server>Server:XFree86(ati)</server>
              <bus>PCI</bus>
              <memory>256</memory>
              <id>268584786</id>
              <ramdac/>
              <clockchip/>
            </videocard>
        </videocards>
        <mouses>
            <mouse>
              <vendor>Unknown</vendor>
              <model>Unknown</model>
              <device>/dev/psaux</device>
              <module>ignore</module>
              <bus>PS/2</bus>
```

```
            </mouse>
         </mouses>
         <netcards>
            <netcard>

              <vendor>3Com Corporation</vendor>
              <model>3c980-TX 10/100baseTX NIC [Python-T]</model>
              <module>3c59x</module>
              <num>0</num>
              <bus>PCI</bus>
              <id>280467461</id>
            </netcard>
         </netcards>
         <disks>
            <disk>
              <vendor>SEAGATE</vendor>
              <model>ST318406LW</model>
              <device>/dev/sda</device>
              <bus>SCSI</bus>
              <cylinders>2231</cylinders>
              <heads>255</heads>
              <sectors>63</sectors>
            </disk>
         </disks>
         <bridges>
            <bridge>

              <vendor>Advanced Micro Devices [AMD]</vendor>
              <model>AMD 765 [Viper] ISA</model>
              <module>ignore</module>
              <bus>PCI</bus>
            </bridge>
         </bridges>
         <others>
            <other>

              <vendor>Advanced Micro Devices [AMD]</vendor>
              <model>Unknown</model>
              <device>Not Available</device>
              <module>Unknown</module>
              <bus>PCI</bus>
              <id>270692364</id>
            </other>
         </others>
      </hardware>
</AUTOMATICALLY_GENERATED_XML>
```

Haciendo un ejercicio de imaginación veremos para qué cosas podríamos utilizar el XML anterior, teniendo el modelado de datos creado. A nivel de una sola máquina podemos saber cuál es el *hardware* que tiene, su potencia (sin preocuparnos de dónde obtener esos datos en /proc). Guardando lo que tiene el sistema y detectando qué es lo que hay nuevo en el arranque se pueden lanzar asistentes de configuración o, por lo menos, detectar de manera sencilla qué *hardware* ha cambiado.

En un entorno distribuido podemos obtener la información de cualquier nodo para poder tomar decisiónes basadas sobre la potencia de las máquinas, su memoria o cualquier otro componente *hardware* que pudiese tener el equipo.

Por otro lado, añadir más información a este esquema es tan sencillo como añadir nuevas etiquetas. Generar un HTML o XHTML para mostrar en un navegador de manera visual bonita o validarlo es sencillo. Para almacenarlo se puede utilizar una base de datos, compresión, o cualquier otra cosa para trabajar de una manera comoda.

Hay que pensar también que aunque estemos hablando siempre de sistemas de GNU/Linux, toda esta tecnología se puede portar a otros sistemas libres o no libres para crea un estándar. Así esta información no sólo nos la podrían aportar nuestras máquinas GNU/Linux, sino también máquinas *BSD, otros UNIX comerciales o quizás incluso Windows. Esto nos permitiría aislarnos del sistema operativo, por ejemplo no dependiendo de /proc para conocer las capacidades de nuestro procesador sino teniendo un XML con esa información y poco nos importa cómo se haya conseguido. No enumeraremos aquí las ventajas que una abstracción así puede tener, simplemente queremos dejar constancia de que aunque nuestra plataforma favorita sea GNU/Linux, el proyecto podría ser de utilidad a otras comunidades, lo que lo hace todavía más interesante.

Desventajas del XML.

Como suele ocurrir con cualquier tecnología, no sólo existen ventajas, sino que también existen varios problemas; para mucha gente un fichero XML no es tan intuitivo de leer a simple vista como un fichero de texto plano. El problema más grave que hemos encontrado es que cada biblioteca que trabaja sobre XML requiere por parte del programador un conocimiento profundo de representacion DOM, nodos y otras estructuras.

También sabemos que, para la mayor parte de los datos a modelar, el tiempo invertido en conocer cómo funciona toda la tecnología es demasiado compleja y requiere de un tiempo de aprendizaje excesivo. Para pequeñas aplicaciones parece que no tiene sentido un sistema tan complejo.

Creemos que para muchos programadores una sintaxis similar a las de los lenguajes de programación imperativos es suficiente solución, de manera que hemos desarrollado bibliotecas y programas donde se abstrae el conocimiento sobre los nodos DOM. Un ejemplo es el método para acceder al modelo del primer CDROM de la máquina en el XML de *hardware* anterior mostrado:

```
hardware.cdroms.cdrom[0].model
```

y sabemos que las tarjetas de red estan bajo el dominio de :

```
hardware.netcards
```

Lo que supone que si entramos en ese dominio podemos acceder a las tarjetas quitando el camino completo y utilizando rutas relativas, de tal manera que para cargar el módulo de todas las tarjetas de red bastará con algo como colocarse en el dominio *hardware.netcards*, preguntar cuantas tarjetas hay y por cada tarjeta leer su *netcard[n].module* y efectuar la pertinente carga del módulo sobre el sistema.

Todo esto son ejemplos de cómo se puede trabajar con XML para salvar el problema que nos ocupa. De este modo el programador sólo tiene que acceder al árbol de una manera muy similar a como accedería a una estructura con vectores de un lenguaje imperativo como C.

Este método no tiene por quí ser perfecto, ni tan siquiera bueno, pero queremos dejar constancía de que se pueden hacer aproximaciones más abstractas al manejo de XML que accesos directos a nodos o recibir eventos (DOM y SAX). Para aplicaciones donde prima la velocidad seguramente estos sean los mejores métodos, pero donde lo que se pretende es una baja complejidad de un sistema, estamos seguros de que existen formas más sencillas para acceder a XML.

INTERFACES GRÁFICOS.

La importancia de los UI.

¿Por qué hemos dado tanta importancia a los UIs en esta ponencia hasta ahora? Una vez explicado el esquema de desarrollo, el tema de los UIs ¿no queda a un lado respecto a cómo gestionar la información? Cada vez se da más importancia al aspecto gráfico de las máquinas GNU/Linux.

La integración de los escritorios, así como todas las aplicaciones que funcionan sobre las X deben ser intuitivas y sencillas de manejar. Con este esquema conseguimos que la gente que se dedica a hacer interfaces gráficos de usuario pueda dedicarse plenamente al aspecto gráfico, que es lo que acabará viendo el usuario final.

Las ventajas en el escritorio.

El trabajo sobre el XML no sólo afecta a los UIs. Los escritorios se deberían ver afectados por ello. Los principales escritorios libres poseen un centro de control del sistema. Estos centros de control han evoluciónado y siguen haciéndolo. Al principio, en estos centros de control sólo se modificaba el aspecto gráfico del escritorio, pero poco a poco se están introduciendo en ellos herramientas para la configuración de aspectos importantes en el sistema: la red, el arranque, e incluso el *kernel* (como es el caso de *kcontrol*).

Este esquema de trabajo presenta una clara ventaja para los escritorios, ya que los desarrolladores de escritorio sólo tienen que preocuparse de cómo integrar sus programas, sin tener que saber necesariamente cómo funcionan los *netlink sockets* para configurar la red o similares y con la ventaja de que cualquier añadido en las bibliotecas de configuración podrá ser aplicado directamente sobre los entornos gráficos del escritorio.

Trabajando con XML.

El trabajo por parte de los desarrolladores de UIs con XML debe ser sencillo. Como comentamos en el apartado anterior, los desarrolladores de UIs no tienen por qué conocer profundamente todas las posiblidades de XML o una biblioteca de XML completa. Para ellos bastaría con que la generacion de un XML con cierta forma, que se valide con un DTD, sea sencilla. Es más, estos XML la mayoría de las veces son accedidos o modificados desde *scripts*.

Para solucionar el problema de cómo un *script* accede a un XML sin que la lectura se complique, hemos desarrollado el programa *xmlManage*, del que hablaremos en la sección de herramientas. De momento decir que *XmlManage* permite trabajar sobre el XML con una sintaxis parecida a la de nuestra biblioteca, lo que supone que una vez conocida una de las dos herramientas la otra es casi inmediata.

HERRAMIENTAS.

Problemas con el XML.

Como ya hemos dicho, si de cada una de las bibliotecas que gestiona los datos en XML e interfaces gráficos tenemos que conocer cómo movernos por nodos y en general el modelo de objeto de documentos, dependendiendo además de la complejidad de las bibliotecas que se utilicen la dificultad se dispara.

Otro hecho es que en la mayoría de los casos el trabajo necesario sobre XML es relativamente simple, con lo que una biblioteca compleja lo único que haría sería entorpecer al desarrollador de los interfaces y bibliotecas. En esta sección, queremos mostrar las herramientas que hemos desarrollado y ejemplos de cómo las utilizamos.

La principal herramienta desarrollada, y sobre la que seguimos trabajando, es *libbslxml*. Esta biblioteca está actualmente creada sobre *tinyQT*, aunque en el futuro tenemos pensado portarla a *libxml2*. Toda la distribución que realizamos utilizaba QT, así que, en nuestro caso, la dependencia no era ningún problema. En cualquier caso *libbslxml* compilada con *tiniQT* en estático no pasa del megabyte, lo que supone que se podría hacer prácticamente autónoma.

Herramientas para proceso de aprendizaje rápido.

La idea de acceso de manera simple a datos para programadores sobre XML es un tema más complejo de lo que parece a primera vista. La cantidad de estructuras que se utilizan para programación hacen que una API completa, sin perder la limpieza, sea muy difícil de realizar.

Nuestra primera intención ha sido crear un modelo de acceso a los datos que emule estructuras y vectores de los lenguajes imperativos mayoritarios, de manera que se consiga un periodo de aprendizaje de las funciones básicas casi nulo y un periodo de aprendizaje de las funciones avanzadas de una hora, en el caso de contar con un programador acostumbrado a lenguajes de programación imperativa.

Libbslxml.

Libbslxml es, como hemos dicho, la implementaciín de la biblioteca que permite el acceso simple a los datos de un XML. Está implementada en C++, pero crear una biblioteca similar en C sería sencillo. Dada la actual dependencia de Qt o TinyQt, la mayoría de las salidas proveen un QString, lo que quizá en la mayoria de los casos no sea lo más deseable. *Libbslxml* permite las siguientes funcionalidades:

- Acceso de lectura/escritura de datos a la manera de estructuras y vectores de lenguajes imperativos mayoritarios.
- Ubicación a nivel de rutas relativas o rutas absolutas.
- Posibilidad de crear y leer atributos de las etiquetas.
- Funciones de busqueda sobre el XML a nivel de contenidos e información.
- Trabajo con Nodos y Atributos.
- Formateo del XML en texto.
- Funciones de creación y borrado de dominios y etiquetas.
- Funciones de copia de secciones de un documento a otro.
- Funciones que permiten contar etiquetas dentro de un dominio.

Como ejemplo, este código:

```
XmlConfig *xml = new XmlConfig();
xml->createElementSetDomain("hardware.cpus.cpu");
xml->createElement("id","0");
xml->createElement("frequency","1546");
xml->delDomain();
xml->createElementSetDomain("hardware.cpus.cpu");
xml->createElement("id","1");
xml->createElement("frequency","2000");
xml->delDomain();
xml->save("/tmp/prueba.xml");
```

Generaría el siguiente fragmento de xml:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<AUTOMATICALLY_GENERATED_XML>
    <hardware>
        <cpus>
            <cpu>
              <id>0</id>
              <frequency>1546</frequency>
            </cpu>
            <cpu>
```

```
            <id>1</id>
            <frequency>2000</frequency>
          </cpu>
        </cpus>
      </hardware>
    </AUTOMATICALLY_GENERATED_XML>
```

Y la lectura sería más o menos igual de simple, por ejemplo

```
xml->delDomain();
if (!xml->setDomain("hardware.cpus"){
cerr << "no existe dominio de CPUS" << endl;
)
for(int i=0;i<xml->howManyTags("cpu");i++){
cout << "Cpu numero " << endl;
cout << (xml->readString("cpu["+QString::number(i)+"].id")).latin1() ;
cout << (xml->readString("cpu["+QString::number(i)+"].frequency")).latin1() << endl ;
}
```

xmlmanage.

En muchos de los casos tener una biblioteca simple no es suficientemente flexible. Por ejemplo, para escribir partes de configuración en un XML sacada directamente de */proc*, o diálogos sencillos en *ncurses* que quieran modificar los datos, un *script* es lo más sencillo y lo más utilizado en UNIX. El problema es que trabajar con ese XML debería ser suficientemente sencillo.

Para trabajar sobre esos datos de manera similar a como lo hacemos con *libbslxml*, pero haciendo uso de *scripts*, hemos creado *xmlmanage*. Se trata de un programa de línea de comando que permite utilizar de una manera flexible, dentro de las posibilidades de la línea de comando, todas las funcionalidades que se proveen en *libbslxml*.

Esto nos evita la necesidad de usar *awk* o alguna otra forma de tratamiento de texto para manejar el XML y nos asegura que el fiche XML estará siempre en un estado correcto.

Por ejemplo, para crear el XML enunciado más arriba habría bastado con:

```
xmlmanage w /tmp/prueba.xml hardware.cpus
xmlmanage --write /tmp/prueba.xml hardware.cpus.cpu.id 0
xmlmanage w /tmp/prueba.xml hardware.cpus.cpu.frequency 1546
xmlmanage w /tmp/prueba.xml hardware.cpus.cpu[1].id 1
xmlmanage w /tmp/prueba.xml hardware.cpus.cpu[1].frequency 2000
```

Y para leerlo algo similar a:

```
xmlmanage --list nodes /tmp/prueba.xml hardware.cpus cpu
```

Éste daría como salida dos líneas con los nodos *cpu* que podrían recogerse con *wc -l*, para saber que son 2 y luego pasar a hacer la lectura.

```
xmlmanage --read /tmp/prueba.xml hardware.cpus.cpu[0].id
xmlmanage --read /tmp/prueba.xml hardware.cpus.cpu[0].frequency
```

EJEMPLO.

Como explicamos al principio de la ponencia, el esquema de desarrollo descrito hasta ahora no es una utopía. Las aplicaciones que hemos ido creando sobre la distribución GNU/Linux fueron diseñadas así en parte para poder implementar ciertos requisitos del sistema y en parte para demostrar que la utilización de este esquema de desarrollo puede ser un comienzo para herramientas de administración distribuidas para grandes sistemas GNU/Linux.

Distribución.

La distribución GNU/Linux esta basada integramente en Debian, la explicación del porqué, la motivación y tecnología de cómo fue hecha va más allá de los límites de esta ponencia. Sólo vamos a explicar qué ventajas hemos obtenido del modelo mostrado.

La mayoría de las variables que modifican las instalaciones de Debian estaban en listas contenidas en *scripts* y similares. Dispersas en diversos ficheros por todo el sistema de instalación. Hemos reescrito todo ese código en C++ para que sea más sencillo de mantener, y las variables en listas así como el comportamiento y aspecto gráfico de la instalación sean leídos de varios XML, cada uno con su propia función.

De este modo, hacer personalizaciones sólo supone modificar un XML. Otra ventaja es que la distribución se separa en dos partes.

GUI: Esta parte no es más que un interfaz gráfico que completa un XML con la configuración que va eligiendo el usuario, mediante unas pantallas lo más amigables posibles.

Backend: Esta parte es un programa llamado *coolbluebackend*, que interpreta el XML para instalar, particionar y configurar todo tal y como especifica dicho XML. Esto nos permite :

- Configurar el XML previamente, por ejemplo vía *web* y crear distribuciones personalizadas.
- Configurar en red cualquier máquina. Sólo hay que pasar el XML a través de la red a un programa que contenga el interfaz gráfico y cuando el XML esté completo, cederselo al *coolbluebackend* de la máquina donde se desea instalar.

CoolDrive.

CoolDrive es el particionador que se utiliza en la distribución anterior. Existe una biblioteca llamada *libbslparted*, que es capaz de:

- Obtener un modelado de datos XML con la representación de los discos de una máquina.
- Generar un XML con las operaciones que deben realizarse sobre una máquina, sólo haciendo uso de dos documentos XML que contengan el estado inicial y final de la máquina.

- Particionar un disco haciendo uso de un XML con operaciones contenidas en el XML, o con las generadas por un XML de modelado de datos inicial y final del disco.

La biblioteca llama en última instancia a una mezcla de funciones propias y funciones de la *libparted*. Una vez desarrollamos la biblioteca, creamos un programa que representaba gráficamente la información de los discos.

Durante una de las mañanas del anterior Hispalinux, programamos toda la funcionalidad de red necesaria para que el particionador trabajase de manera remota y todo ello nos llevó apenas 2 o 3 horas, lo cual da una buena idea de la potencia que tiene este esquema para el trabajo en red.

Teniendo un demonio que nos permitiera recibir la información de almacenamiento de todas las máquinas, podríamos elegir una de ellas y presentar el interfaz gráfico para particionar ese disco. Mandar la configuración de cómo debe particionarse, junto con el XML se cede la configuración para compartir mediante SAMBA o NFS ese espacio de almacenamiento y enviar un XML a todas las máquinas para que modifiquen su *fstab* y traten de montar el sistema remoto...

Esto que parece tan sencillo, lo es. Y más de la mitad de este esquema está actualmente hecho. Para hacer los esquemas funcionales en GNOME sólo habría que hacer los UIs, ya desarrollados en KDE, a causa de los ejemplos comentados.

Este particionador no pretende quedarse ahí. La biblioteca manejará LVM y RAID del mismo modo y de igual manera el sistema de gestión gráfica será potente, sencillo e intuitivo.

GUI-XML-LIBBSLNET-TC, IP-NETLINK SOCKET

Estamos utilizando un esquema idéntico al del particionador en la biblioteca de red. Actualmente sólo soporta *iproute* y toda su sintaxis, pero en el futuro implementaremos toda la tecnologia de tcng (*tc next generation*), para controlar la calidad de servicio y reglas de *iptables*.

Claramente el caso de la red es un ejemplo de los problemas de los que hablamos al principio de la conferencia. Actualmente la configuración de red de nuestros GNU/Linux depende de la distribución que utilicemos y de todos los ficheros que hay para configurar direcciones, nivel de enlace, encaminamiento, cortafuegos, redes privadas virtuales, calidad de servicio de red, configuración de DNSs a utilizar, etc.

Todo lo cual repercute en manejar muchos ficheros, y hace que la gestión del sistema sea innecesariamente lenta. Con este esquema se intenta solventar este problema. Actualmente la biblioteca tiene unas opciones similares a las del particionador de *coolblue*.

¿Y AHORA, QUÉ?

Ahora queda que los desarrolladores vean la potencia de estas ideas en UNIX. Uno de los grandes problemas para solucionar la administración de oficinas, aulas o cualquier conjunto de máquinas UNIX, es que el administrador puede tocar tantas cosas que lo que era ventaja en un sólo ordenador se convierte en desventaja por el tiempo invertido en muchos ordenadores. Por otro lado hay

miles y miles de líneas de codigo reescrito en aplicaciones de configuración que se evitarían teniendo un modelado de configuración estándar en el /etc.

Por último, y lo más importante, si queremos tener sistemas de siguiente generación donde cuando hablamos de sistema hablamos de un conjunto de máquinas que trabajan de manera conjunta, donde un sólo administrador es capaz de aplicar cambios estándar a todas ellas y, sobre todo, donde podemos aplicar analizadores de datos que tomen decisiones y las apliquen sobre el sistema de manera que el trabajo del administrador quede relegado a asentir en el caso de que el sistema acierte en sus decisiónes y a "pasar a modo manual" en el caso contrario, necesitamos un modelado de datos de configuración. Es decir, la parte de ciencia ficción que, como hemos mostrado, tiene más parte de ciencia de lo que parecía al iniciar la investigación, y una parte de ficción que tenemos que abordar entre todos.

De momento la ciencia ficcion no sirve y solo queda avanzar paso a paso. Con una de las posibles soluciones, hemos demostrado que se puede hacer y es compatible con la configuración actual, a través de FAM para modificar nuestro XML, así como nuestros ficheros de configuración reales, hasta que los desarrolladores tomen la decisión de migrar sus ficheros de configuración y bibliotecas a un estándar nuevo y mejor.

Para el sistema de acceso a la configuración *CORBA* o *Dcop* junto con sistemas de autenticación *kerberos* o un sistema similar de registro y acceso remoto a objetos sería ideal para hacer que el acceso a las funciones que nos proveen otras máquinas pueda ser utilizado no sólo a nivel de aplicación de usuario, sino también a nivel de configuración de sistema, de manera que unos simples *scripts* escritos por administradores hagan el trabajo de manera simple en cientos de máquinas.

Y sobre todo, la evolución del esquema. Tomado a pequeña escala, a nosotros nos ha reportado grandes ventajas. Tomado a gran escala, por ejemplo, en un proyecto como puede ser Debian GNU/Linux y herramienas adicionales, el esquema puede representar un nuevo paso en administración y en computación un paso hacia la posibilidad de sistemas distribuidos reales.