Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid

Avenida Complutense nº 30 28040 Madrid

Diseño de una interfaz gráfica para la herramienta de gestión de escenarios virtuales VNX

Trabajo Fin de Máster

Iván Rodríguez Lahera
ivan.rodriguezl@alumnos.upm.es

Madrid, Julio 2012

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO DE UNA INTERFAZ GRÁFICA PARA LA HERRAMIENTA DE GESTIÓN DE ESCENARIOS VIRTUALES VNX.

Autor
Iván Rodríguez Lahera.

Director
David Fernández Cambronero.

Departamento de Ingeniería de Sistemas Telemáticos

2012

DEDICATORIA

Este trabajo está dedicado a mi familia por su apoyo incondicional.

AGRADECIMIENTOS

Muchas fueron las personas que contribuyeron a la realización de este trabajo, brindando no sólo sus experiencias, sino su apoyo personal y emocional.

Quisiera agradecerle al tutor de este proyecto David Fernández Cambronero por su iniciativa y apoyo durante todos estos meses de trabajo.

A los creadores y colaboradores de las herramientas de virtualización VNX, VNUML, y VNUMLGUI.

A mis padres, y demás familiares que sin su ayuda hubiera sido imposible la realización del trabajo.

A mis compañeros de aula y amigos que de una forma u otra han ayudado a realizar este trabajo.

RESUMEN

La virtualización es una de las herramientas más importantes para el análisis del comportamiento de las redes de comunicaciones, lo cual resulta un beneficio directo en la reducción del hardware necesitado y su respectivo costo. En este trabajo se realiza el diseño de una interfaz gráfica para la herramienta de gestión de escenarios virtuales de redes VNX, para ello se estructurará en las siguientes partes: La primera parte comenzará con las funcionalidades y características de VNX. La segunda consistirá en un estudio de la construcción de interfaces gráficas mediante la biblioteca GTK y el lenguaje de programación Perl, se describirá además el uso de GLADE para el diseño gráfico de aplicaciones GTK. La tercera será dedicada a la construcción de la interfaz gráfica VNXGUI, en la cual primeramente se partirá de las diferencias entre VNX y VNUML, y se abordará el estudio de la interfaz gráfica VNUMLGUI para la versión anterior de VNX (VNUML). Y finalmente se simulará un escenario de prueba para el nuevo diseño de VNXGUI.

ABSTRACT

Virtualization is one of the most important tools for analyzing the behavior of communications networks, bringing a direct benefit in the reduction of hardware needed, and associated costs. In this work we make the design of a GUI tool for managing virtual network environments VNX, for that will be structured into the following parts: The first part will begin with the functionality and features of VNX. The second will study the construction of graphical interfaces using the GTK library and the programming language Perl, also will be described the use of GLADE for the graphic design of GTK applications. The third will be dedicated to building the graphical user interface VNXGUI. It begins with the differences between VNX and VNUML, followed by the study of the graphical user interface VNUMLGUI for the previous version of VNX (VNUML). And finally will be simulated a test scenario for the new design of graphical interface VNXGUI.

INDICE

INTRODUCCION

La virtualización se ha convertido en una técnica muy importante en la actualidad, su principal ventaja radica en que si puedes virtualizar un número de sistemas en un solo equipo, se logrará un ahorro de costes, tanto de infraestructura como esfuerzo invertido en su gestión y configuración. No es comparable el gasto que habría de realizarse en una simulación con entorno real que hacerla en un único pc.

En particular este trabajo se centra en la herramienta de virtualización de escenarios de redes: Virtual Networks over Linux (VNX). VNX es la nueva versión de Virtual Network User Mode Linux (VNUML). VNX es una herramienta de virtualización diseñada para crear complejos escenarios de redes, está basada: en el software de virtualización UML (user mode linux), en la colección de software Libvirt que provee una forma conveniente de manejar maquinas virtuales y otras funcionalidades de virtualización, en la solución de virtualización KVM, en los emuladores de routers: Dynamips para cisco y Olive para Juniper.

Su principal ventaja es que aporta un entorno de simulación real y potente, capaz de involucrar docenas de nodos evitando el gasto que supondría el traslado a un entorno real.

VNX esta constituido por dos partes fundamentales:

- Un lenguaje XML para la descripción del escenario virtual de red.
- El programa VNX que construye y gestiona el escenario virtual en una máquina Linux.

Con lo cual la elaboración de un escenario VNX, obliga al usuario a redactar un fichero XML, en el cual irá describiendo elemento por elemento, sus características mediante etiquetas y atributos.

La idea principal de este trabajo es la creación de una interfaz gráfica que permita la construcción y ejecución de los escenarios de simulación de VNX, o sea que de forma visual sean insertados los elementos que conforman el escenario y automáticamente genere el código XML, para luego suministrárselo a VNX. Esta herramienta gráfica estará basada en GTK – Perl y será nombrada VNXGUI. Para ello se partirá de la herramienta ya construida VNUMLGUI, encontrada en el sitio sourceforge.

Planteamiento del problema

El problema fundamental radica en que la herramienta gráfica desde la que se partirá para la programación de la nueva interfaz gráfica está bastante desfasada, y existen numerosos cambios realizados en la sintaxis del código XML, esto impide por supuesto la carga de escenarios hechos en VNX, en VNUMLGUI, así como la creación de

nuevos escenarios de simulación por esta herramienta gráfica para posteriormente suministrárselos a VNX.

Objetivos

Este problema hace que se planteen los siguientes objetivos en la realización de este proyecto:

- Análisis y comprensión de la herramienta de virtualización VNX.
- Estudio del lenguaje Perl junto a la librería gráfica GTK para la construcción de interfaces gráficas.
- Entendimiento y modificación del código de programación de la herramienta gráfica VNUMLGUI.
- Diseño de un escenario virtual con la nueva herramienta VNXGUI.

Método experimental

Para ello se estructurará un modelo sobre cómo se llevará a cabo la elaboración de los diferentes capítulos en este trabajo:

1. El primer capítulo abordará un estado del arte de la herramienta VNX, donde se tratará temas como: una revisión de las tecnologías empleadas en VNX, un análisis del funcionamiento de VNX y la comprensión del lenguaje empleado en la creación de escenarios VNX (código XML).
2. El segundo capítulo se centrará en el estudio de la librería gráfica GTK, realizando un análisis de sus principales componentes, junto al lenguaje de programación Perl. Además se dará una breve explicación del uso de GLADE para la construcción de interfaces gráficas, junto a un pequeño ejemplo.
3. Finalmente el tercer capítulo tratará en cuestión la realización de la interfaz gráfica VNXGUI. Primeramente se comenzará con una descripción de cuales son los cambios que se han realizado en VNX con respecto a VNUML. Luego se mostrará una explicación del funcionamiento VNUMLGUI, en cuanto a sus funciones principales, para posteriormente continuar con los cambios realizados en el código del programa. Y para terminar el capítulo se realizará una breve explicación de la simulación de un escenario de pruebas en VNXGUI.

Planificación

Para llevar a cabo las tareas planteadas, se realizará una planificación, lo que permitirá ser sistemáticos en la realización de este trabajo. La siguiente tabla muestra dicha planificación:

Nombre de tarea	Duración	Comienzo	Fin
Inicio de Trabajo de Fin de Master	**0 días**	**sáb 19/11/11**	**sáb 19/11/11**
Estudio de estado del arte: VNX	**29 días**	**sáb 19/11/11**	**jue 29/12/11**
Revisión de las tecnologías empleadas en VNX	1 ms	sáb 19/11/11	jue 15/12/11
Análisis del funcionamiento de VNX	0,5 mss	vie 16/12/11	jue 29/12/11
Comprensión de lenguaje VNX (Código XML)	0,5 mss	vie 16/12/11	jue 29/12/11
Estudio de Lenguaje GTK-Perl	**20 días**	**lun 02/01/12**	**vie 27/01/12**
Investigación de los componentes y funcionamiento de GTK	0,5 mss	lun 02/01/12	vie 13/01/12
Revisión de la Herramienta GLADE	**10 días**	**lun 16/01/12**	**vie 27/01/12**
Construccion de interfaz basado en GLADE	0,5 mss	lun 16/01/12	vie 27/01/12
Desarrollo de la Interfaz Gráfica VNXGUI	**120 días**	**lun 30/01/12**	**vie 13/07/12**
Análisis de diferencias entre herramientas previas y actuales: VNUML y VNX	1 ms	lun 30/01/12	vie 24/02/12
Aprendizaje de la herramienta VNUMLGUI	0,5 mss	lun 27/02/12	vie 09/03/12
Instalación de SW requerido	10 días	lun 12/03/12	vie 23/03/12
Programación de la Interfaz: VNXGUI	**60 días**	**lun 26/03/12**	**vie 15/06/12**
Adaptación de códigos perl en interfaz VNUMLGUI	3 mss	lun 26/03/12	vie 15/06/12
Simulación de escenario de Pruebas	1 ms	lun 18/06/12	vie 13/07/12
Conclusiones	5 días	lun 16/07/12	vie 20/07/12
Fin	0 días	lun 23/07/12	lun 23/07/12

Tabla 1. Planificación del trabajo final de Master.

Capítulo 1: Virtual Networks over linuX

La virtualización es un proceso complejo que se basa principalmente en montar un sistema operativo por encima del que usamos normalmente.

Varias máquinas virtuales comparten recursos de hardware sin interferir entre sí de modo que se pueden ejecutar simultáneamente y de forma segura varios sistemas operativos y aplicaciones en un único ordenador.

Virtual Networks over Linux (VNX) es una herramienta de virtualización basada en código abierto, diseñada para rápidamente definir y testear escenarios de simulación de redes complejos.

Su objetivo es ayudar en el testeo de aplicaciones y servicios de red sobre complejos bancos de prueba, formado por varios nodos y redes dentro de una máquina Linux, sin la participación de la complejidad de inversión y gestión necesitada para crearla usando equipamiento real.

El comienzo de VNX fue con su versión anterior Virtual Network User Mode Linux (VNUML) desarrollada a finales del 2002, VNUML ha sido ampliamente usada en varios campos relacionados con las redes y ciencias de la computación.

VNUML fue originalmente desarrollada por el Departamento de Ingeniería de Sistemas Telemáticos (DIT) de la Universidad Politécnica de Madrid en España, este software es lanzado bajo la licencia pública GNU (1).

VNX, que mantiene la simulación de escenarios basados en el software de virtualización User Mode Linux (UML), pero además brinda nuevas funcionalidades que superan las limitaciones más importantes de la herramienta VNUML. Estas nuevas funcionalidades son:

- Integración de nuevas plataformas de virtualización para permitir a las maquinas virtuales ejecutar otros sistemas operativos como Windows, FreeBSD. VNX usa libvirt para interactuar con las capacidades de virtualización, además integra dynamips para permitir la emulación de routers cisco y la plataforma de virtualización de routers Olive, para la emulación de routers Juniper.

- Gestión individual de las máquinas virtuales.

- Autoconfiguración y capacidades de ejecución de comandos para varios sistemas operativos: Linux, FreeBSD y Windows (XP and 7).

Las arquitecturas de enrutamiento, plataformas de servicios multimedia IP y seguridad lógica son algunas actividades de investigación y desarrollo de aplicaciones que se realizan sobre esta herramienta. Adicionalmente, VNX es usada en educación superior con el fin de construir los laboratorios de formación para los estudiantes.

1.1 Tecnologías usadas por VNX

VNX se apoya en varias plataformas de virtualización para la creación de los escenarios de simulación. Como su versión anterior VNUML, VNX sigue utilizando el software de virtualización UML, pero además integra libvirt, dynamips y olive. En este epígrafe, se realizará una breve explicación de esas tecnologías con el fin de un mejor entendimiento de la herramienta VNX (2).

1.1.1 UML

User-Mode Linux es un modo seguro de ejecutar procesos dentro del Sistema Operativo GNU/Linux llegando incluso a permitir la ejecución de distintas versiones de GNU/Linux dentro de una misma máquina. Está diseñado para permitir a desarrolladores experimentar distintas versiones de GNU/Linux sin poner en peligro la instalación del Sistema Operativo principal del ordenador. User-Mode Linux nos ofrece una máquina virtual que puede llegar a tener incluso más recursos hardware y software que el ordenador físico que la está ejecutando (3).

UML se diferencia de otras tecnologías de virtualización, en ser más, un sistema operativo virtual, en lugar de una máquina virtual. Tecnologías tales como VMWare son en realidad maquinas virtuales. Ellas emulan una plataforma física, desde la CPU, hasta los periféricos, lo cual tiene la ventaja de que normalmente cualquier sistema operativo (SO) puede ejecutarse. Por el contrario, UML solo puede ser una máquina virtual Linux (Linux guest), por otro lado al ser más un sistema operativo virtual en lugar de una máquina virtual, permite interactuar más plenamente con el sistema operativo anfitrión, lo cual tiene otras ventajas.

Otras tecnologías de virtualización como Xen, BSD jail, Solaris zones y chroot están integradas en el SO, lo contrario de UML, que se ejecuta en un proceso. Esto le da la ventaja a UML, de ser independiente de la versión del SO del host, a costa de algo de rendimiento. Sin embargo este aumento del rendimiento puede ser compensado por la ganancia de espacio en el SO del host.

La figura 1.1 muestra la relación entre una instancia UML, el kernel del host y los procesos UML. Para el kernel del host, la instancia UML es un proceso normal. Para los procesos UML, la instancia UML es un kernel. Los procesos interactúan con el kernel haciendo llamadas de sistema, o sea como otros procesos en el host, UML realiza llamadas de sistema, al kernel del host, para su funcionamiento. A diferencia de los demás procesos del host, UML tiene su propia interfaz de llamadas de sistema, para el uso de sus propios procesos. Esta es la fuente de la dualidad de UML, realiza

llamadas de sistema al host, lo que lo hace un proceso y a su vez implementa llamadas de sistema, para sus propios procesos, convirtiéndolo en un kernel.

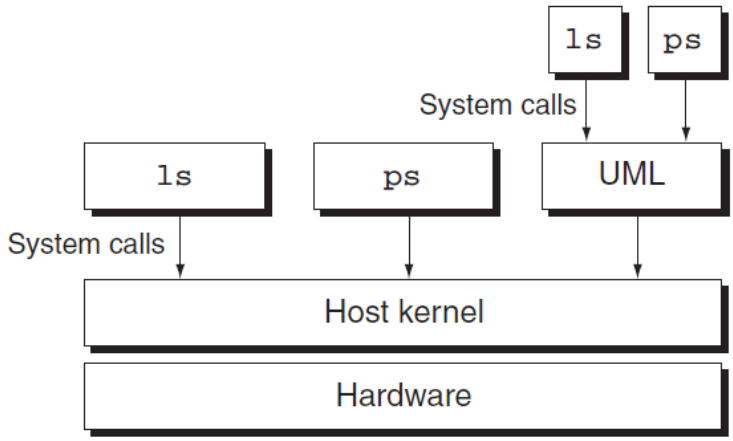


Figura 1.1. UML como proceso y kernel.

1.1.2 Libvirt

Libvirt es una colección de software que provee una forma conveniente de manejar maquinas virtuales y otras funcionalidades de virtualización, como gestión de almacenamiento e interfaces de redes. Estas piezas del software incluyen una librería API, un demonio (libvirt), y una utilidad para la línea de comandos (virsh) (4).

El objetivo principal de libvirt es proveer una forma simple para manejar múltiples proveedores/hipervisor de virtualización diferentes. Por ejemplo, el comando 'virsh list --all' puede utilizarse para listar todas las maquinas virtuales existentes para cualquier hipervisor soportado (KVM, Xen, VMWare ESX, etc). No se necesita aprender las herramientas específicas de cada hipervisor.

Libvirt como muestra la figura 1.2 existe como un conjunto de APIs diseñadas para ser usadas por una aplicación de gestión, a través de un mecanismo específico es capaz de comunicarse con cada hipervisor disponible (5).

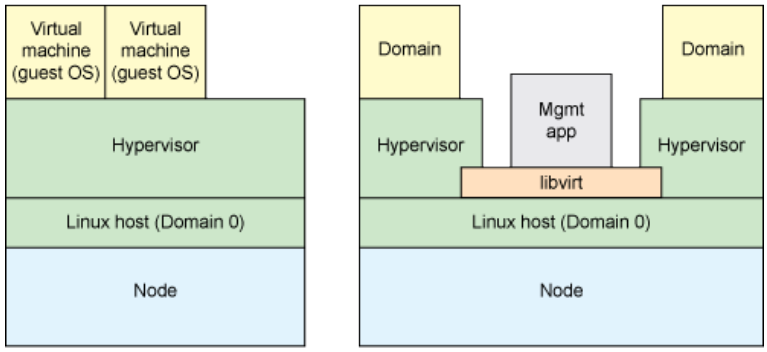


Figura 1.2. Uso del modelo de libvirt.

La terminología en libvirt también es importante, como se nota en la figura anterior libvirt llama al host como nodo y su sistema operativo como domain, por lo que se puede notar que libvirt y sus aplicaciones son ejecutadas en el SO del host (domain0).

Libvirt tiene dos formas distintas de control. En la figura 1.2 se observa que los dominios y aplicaciones de gestión están en el mismo nodo, en este caso las aplicaciones de gestión funcionan a través de libvirt para controlar los dominios locales. La otra forma de control es cuando las aplicaciones de gestión y los dominios están en nodos separados, en ese caso se requiere una comunicación remota, ver figura 1.3.

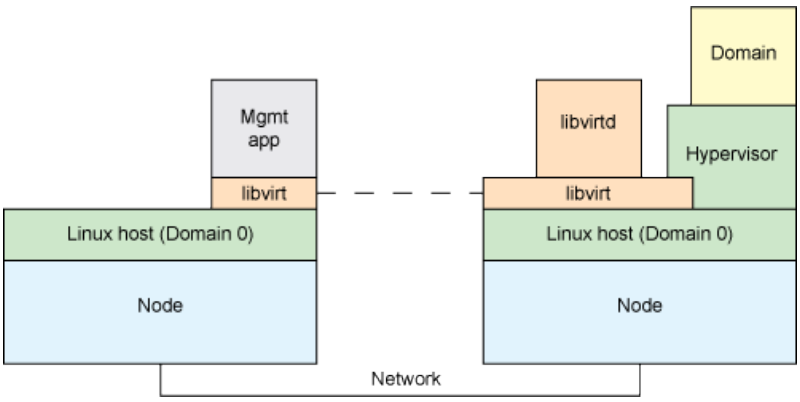


Figura 1.3. Control de hipervisores remotos con libvirtd.

Esta forma utiliza un demonio especial llamado libvirtd, que se ejecuta en el nodo remoto cuando se instala libvirt. Por lo que automáticamente determina los hipervisores locales y configura los drivers para ellos. La aplicación de gestión se comunica a través del libvirt local al libvirtd remoto.

La siguiente tabla muestra un amplio soporte de hipervisores por parte de libvirt:

Hypervisor	Description
Xen	Hypervisor for IA-32, IA-64, and PowerPC 970 architectures
QEMU	Platform emulator for various architectures
Kernel-based Virtual Machine (KVM)	Linux platform emulator
Linux Containers (LXC)	Linux (lightweight) containers for operating system virtualization
OpenVZ	Operating system-level virtualization based on the Linux kernel
VirtualBox	Hypervisor for x86 virtualization
User Mode Linux	Linux platform emulator for various architectures
Test	Test driver for a fake hypervisor
Storage	Storage pool drivers (local disk, network disk, iSCSI volume)

En el caso de VNX, es utilizado libvirt con el hipervisor KVM (de Kernel-based Virtual Machine). KVM es una solución de virtualización para Linux corriendo sobre hardware x86 que hace uso de las extensiones de virtualización por hardware del procesador (Intel VT o AMD-V).

Con KVM, un host Linux puede ejecutar múltiples máquinas virtuales con Linux o Windows sin ninguna modificación necesaria en estos. Cada máquina virtual tiene hardware virtualizado privado: tarjeta de red, disco, adaptador gráfico, etc.

1.1.3 Dynamips

Dynamips es un emulador de routers cisco que emula plataformas hardware 1700, 2600, 3600, 3700, 7200 y ejecuta imágenes IOS estándares. Específicamente en VNX, se utilizan dos imágenes, 3600 y 7200 (6).

Este tipo de emulador es muy útil para:

- Ser usado en plataformas de entrenamiento, con software usado en la vida real. Permite familiarizarse con equipos cisco.
- Testeo y experimentación de características de la IOS de cisco.
- Chequeo rápido de configuraciones para ser desplegados luego en routers reales.

1.1.4 Olive

La plataforma de virtualización de routers Olive, permite emular routers Juniper. Para ello hace uso del sistema operativo FreeBSD, pues JunOS está basado en este SO.

Una vez instalado el JunOS, que es el software que permite la emulación de router Juniper y realizando su apropiada configuración, es posible obtener la IOS de un router Juniper. En este trabajo no entraremos en el detalle de como se realiza la configuración, solo comentar de que la herramienta VNX integra la opción de cargar una máquina virtual basada en freebsd, para la emulación de routers Juniper.

1.2 Funcionamiento VNX

VNX es ejecutada por un host con sistema operativo Linux, esta herramienta puede ser usada por usuarios convencionales, pero cuando es ejecutada por root pueden ser usadas características adicionales. Puede ser instalada en diferentes operativos Linux como Ubuntu, Fedora o CentOS, mediante un archivo .tgz. Claro que para eso se deben realizar configuraciones adicionales en las diferentes tecnologías empleadas por VNX, en este trabajo no se explicará con detalle como se realiza su instalación para ellos se puede ir a la web de VNX (1).

VNX tiene las siguientes fases:

Fase de diseño: Primeramente debemos diseñar un escenario de simulación, para eso debemos considerar los siguientes aspectos: el número de maquinas virtuales, la topología (interfaces de red en cada máquina y como son conectadas), que procesos ejecutará cada máquina virtual, etc. Tenemos que tener en cuenta que toda esta simulación correrá sobre la máquina física a la cual le llamamos host. Este host puede formar parte o no de la simulación siempre que la aplicación sea ejecutada por el root, en el caso de un usuario convencional el host no puede formar parte de la simulación.

Fase de implementación: Una vez diseñado el escenario, debemos escribir el código describiendo dicho escenario, esto se hace en un fichero XML, a través de etiquetas y atributos. Existen tres tipos de etiquetas en VNX: etiquetas estructurales, etiquetas de topología y etiquetas de simulación.

Fase de Ejecución: Al escribir el fichero XML, se debe ejecutar el parser de VNX para construir y gestionar el escenario de simulación. El modo ejecución consiste en tres etapas:

- Escenario de construcción: Crea las redes virtuales que interconectarán las maquinas virtuales y el host, y luego reinicia y configura las maquinas virtuales definidas, adicionando direcciones IP, rutas estáticas, o cualquier otro parámetro de red.

• Comandos de ejecución: Unas vez que el escenario ha sido construido, se pueden ejecutar las secuencias de comandos en el. En esta etapa el parser toma los comandos definidos en las etiquetas <exec>, dentro del archivo XML y los ejecuta. Este paso es actualmente opcional, si no necesitas ejecutar ningún comando, entonces no se usa.

• Escenario de liberación: En este paso final, todos los componentes creados son liberados.

El ciclo de vida típico de VNX consiste en ejecutar el paso 1 para crear el escenario, ejecutar el paso 2 para ejecutar los comandos deseados o necesitados y finalmente ejecutar el paso 3 para liberar el escenario.

1.3 Tipos de etiquetas

Existen tres tipos de etiquetas en un archivo VNX (1):

- Etiquetas estructurales, que son usadas para definir la estructura del archivo, organizándolo en diferentes secciones, cada una contiene otras etiquetas. Existen 5 etiquetas de este tipo: <vnx>, <global>, <vm>, <vm_defaults> y <host>. El parser usa estas etiquetas para conocer la estructura del archivo en cualquier modo de ejecución.

- Etiquetas de topología, llevan consigo la semántica relacionada con la topología del escenario, son usadas durante la construcción y liberación del escenario e ignorados durante la ejecución de comandos. Existen 22 etiquetas de este tipo: <ssh_key>, <automac>, <netconfig>, <host_mapping>, <tun_device>, <net>, <bw>, <filesystem>, <mem>, <kernel>, <console>, <mng_if>, <if>, <hostif>, <mac>, <ipv4>, <ipv6>, <physicalif>, <route>, <forwarding>, <user>, y <group>.

- Etiquetas de simulación, llevan consigo la semántica relacionada con la simulación. Son usadas durante la ejecución de comandos, e ignoradas durante la construcción y liberación del escenario. Existen 4 etiquetas de este tipo: <ssh_version>, <basedir>, <filetree> y <exec>.

1.4 Lenguaje VNX:

1.4.1 Introducción

El lenguaje usado para describir las simulaciones en la herramienta VNX, es el lenguaje XML. Como cualquier archivo XML, todos los escenarios VNX comienzan con un lenguaje de esquema para definir la estructura y restricciones de los contenidos, y son las siguientes líneas (2):

<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">

A diferencia de la versión anterior de VNX (VNUML), que usaba DTD (Document Type Definition) para definir la correcta estructura de los elementos de un documento XML, en VNX se utiliza XSD (XML Schema Definition).

De esta forma con un esquema XSD se pueden definir (7):

- Los elementos que aparecen en el documento XML.
- Que elementos son hijos de los elementos principales del documento XML.
- La secuencia en la cual los hijos de los elementos pueden aparecer en el documento XML.
- El numero de hijos de los elementos.
- Cuando un elemento es vacío o puede incluir texto.
- El tipo de datos para los elementos y sus atributos.
- Los valores predeterminados para algunos elementos y atributos.

Si un documento XML no concuerda con la estructura definida del archivo XSD, entonces el documento XML será erróneo.

Los comentarios usados en un archivo XML tienen la siguiente sintaxis:

<!-- este es un comentario -->

Con lo cual, lo que se encuentre dentro de un comentario, será ignorado por el parser de VNX.

Las principales etiquetas de un archivo VNX son ***<global>*** la cual se encarga de describir los elementos globales de un escenario VNX, ***<net>*** que describe las redes

virtuales, ***<vm>*** las especificaciones de las maquinas virtuales, y ***<host>*** que contiene la configuración referente al host. Por ejemplo si se quisiera confeccionar un fichero VNX, que represente a la siguiente topología de red que muestra la gráfica 1.4:

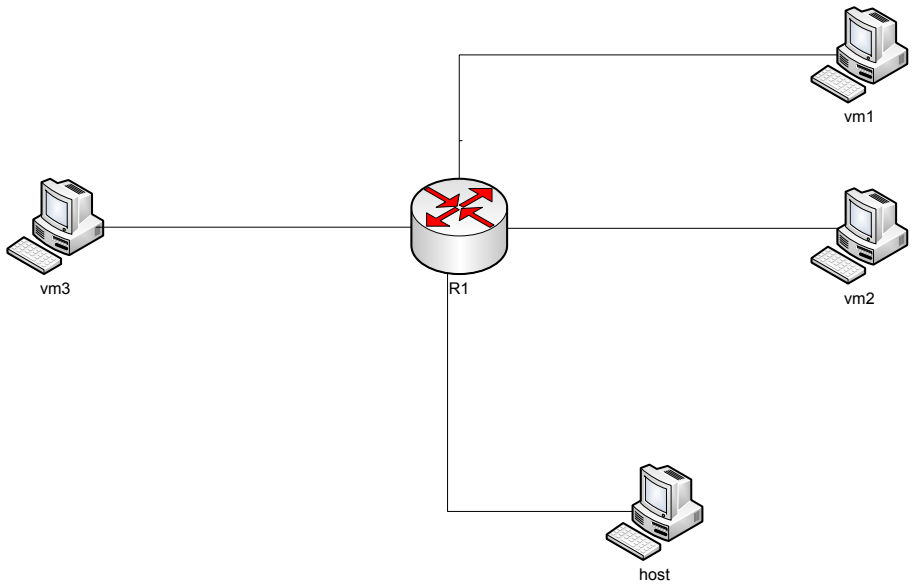


Figura 1.4. Escenario de red.

Donde vm1 y vm2 son máquinas virtuales Linux, las cuales están conectadas junto con un router R1 y el propio host, en una red LAN (Net0) con un direccionamiento 10.1.1.0/24 y a su vez conectamos este router a otra máquina virtual Linux vm3, en una red LAN (Net1) con direccionamiento 192.168.1.0/24.

Tendría que comenzarse con:

```
<?xml version="1.0" encoding="UTF-8"?>
<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
        <global>
                <version>2.0</version>
                <scenario_name>nuevo</scenario_name>
                <automac offset="8"/>
                <vm_mgmt type="private" network="10.9.250.0" mask="24" offset="16">
                        <host_mapping/>
                </vm_mgmt>
        </global>
```

Donde se declaran los atributos globales del escenario, como son la versión a utilizar de VNX, el nombre del escenario de red, la red de gestión, etc.

Luego se tendría que declarar las etiquetas especificando las redes:

```
        <net name="Net0" mode="virtual_bridge" />
        <net name="Net1" mode="virtual_bridge" />
```

Es necesaria siempre la creación de las etiquetas <net> ya que para realizar la conexión entre máquinas virtuales siempre debe estar vinculada a una etiqueta <net>.

Posteriormente se continúa con la definición de las máquinas virtuales que están en el escenario, deben declararse sus características, ya sea memoria, filesystem a utilizar, tipo de máquina, dirección IP, y la red a la que se conecta:

```
        <vm name="vm1" type="libvirt" subtype="kvm" os="linux">
                <filesystem type="cow">/usr/share/vnx/filesystems/root_fs_ubuntu</filesystem>
                <mem>256M</mem>
                <if id="1" net="Net0">
                        <ipv4>10.1.1.1/24</ipv4>
                </if>
        </vm>
        <vm name="vm2" type="libvirt" subtype="kvm" os="linux">
                <filesystem type="cow">/usr/share/vnx/filesystems/root_fs_ubuntu</filesystem>
                <mem>512M</mem>
                <if id="1" net="Net0">
                        <ipv4>10.1.1.2/24</ipv4>
                </if>
        </vm>
        <vm name="R1" type="dynamips" subtype="3600">
                <filesystem type="cow">/usr/share/vnx/filesystems/root_fs_ubuntu</filesystem>
                <mem>96M</mem>
                <if id="1" net="Net0" name="e0/1">
                        <ipv4>10.1.1.3/24</ipv4>
                </if>
                <if id="2" net="Net1" name="e0/2">
                        <ipv4>192.168.1.1/24</ipv4>
                </if>
        </vm>
        <vm name="vm3" type="libvirt" subtype="kvm" os="linux">
                <filesystem type="cow">/usr/share/vnx/filesystems/root_fs_ubuntu</filesystem>
                <mem>256M</mem>
                <if id="1" net="Net1">
                        <ipv4>192.168.1.2/24</ipv4>
                </if>
        </vm>
        <host>
                <hostif net="Net0">
                        <ipv4>10.1.1.3/24</ipv4>
                </hostif>
        </host>
</vnx>
```

Como se observa anteriormente también se declara la configuración de la máquina host donde se esta corriendo toda la virtualización. Este ejemplo permite obtener una idea de cual es la metodología para la creación de un fichero VNX, a continuación se describirá con más detalle el significado de las etiquetas que pueden ser empleadas para la confección de un escenario en VNX.

1.4.2 Etiqueta *<global>*

En esta etiqueta se engloban todas las especificaciones del escenario VNX que no se encontraran dentro de otras etiquetas.

Dentro de la etiqueta ***<global>*** se encuentran las siguientes etiquetas:

<version>

Requerida. Única.

Especifica la versión del lenguaje VNX que se emplea en el escenario. Es usado para realizar un chequeo del VNX parser que lee el documento XML, o sea que los archivos XML versión 2.0 solo pueden ser analizados por la versión 2.0 de VNX.

<scenario_name>

Requerida. Única.

Especifica el nombre del escenario VNX, cada escenario debe tener un nombre diferente.

<ssh_version>

Opcional. Única.

Esta etiqueta es opcional, por defecto el valor es 2. Establece la versión SSH que el parser utilizará cuando se accede a las maquinas virtuales a través de la interfaz de gestión.

Puede tomar dos valores:

1 para SSHv1

2 para SSHv2

<ssh_key>

Opcional. Múltiple.

Cuando es usado, esta etiqueta especifica el nombre del archivo de una llave pública que se encuentre en la máquina host. Cada llave especificada es añadida en la ruta */root/.ssh/authorized_keys*, en el filesystems de todas las maquinas virtuales.

El uso de la autenticación a través de la llave pública elimina la necesidad de password cuando se ejecuten comandos remotos.

Se pueden instalar varias llaves, usando la etiqueta ***<ssh_key>*** tantas veces como se desee. En caso de que se desee instalar la llave en una sola máquina virtual debe ser

ubicado dentro de una etiqueta **<*user*>** con username="root" dentro de la correspondiente máquina virtual **<*vm*>**.

Para generar una llave pública RSA1 (SSHv1) se utiliza el siguiente comando:

ssh-keygen -t rsa1

Para generar una llave pública RSA (SSHv2) se utiliza el siguiente comando:

ssh-keygen -t rsa

Para generar una llave pública DSA (SSHv2) se utiliza el siguiente comando:

ssh-keygen -t dsa

<automac>

Opcional. Única.

Esta etiqueta es usada para generar automáticamente las direcciones MAC de las maquinas virtuales, por lo que no es necesario la utilización de la etiqueta **<*mac*>**. El formato de la dirección MAC es:

fe:fd:0:Z:X:Y

Donde X es el número de la máquina virtual, en el mismo orden en que se especifican las maquinas virtuales en el archivo VNX empezando por 1. El valor Y es el número de la interfaz (atributo id dentro de la etiqueta **<*if*>**) y Z es el valor del atributo opcional *offset*, sino se especificara, se toma el valor 0.

Con esta etiqueta se nota que será imposible que se repita dos veces la misma MAC, en una misma simulación. En el caso de que se ejecuten varias simulaciones concurrentes, se debe especificar diferentes valores de offset para cada escenario, ya que sino es posible que se repita la MAC.

Cuando se utiliza la etiqueta **<*automac*>** y es usado también la etiqueta **<*mac*>** dentro de **<*if*>**, la dirección MAC especificada dentro de la etiqueta **<*mac*>** toma preferencia. En caso de que la etiqueta **<*mac*>** sea usado incorrectamente y este en el rango de las direcciones generadas automáticamente puede dar como resultado el duplicamiento de direcciones MAC.

El uso de **<*automac*>** es recomendado, excepto cuando se necesitan direcciones MAC en específico, por ejemplo cuando se asignan direcciones por DHCP, basadas en direcciones MAC particulares.

Offset puede tomar valores entre 0 y 65535.

<netconfig>

Opcional. Única.

Valores por defecto: *stp="off"* and *promisc="on"*

Esta etiqueta permite configurar la forma en que el parser de VNX configura las interfaces virtuales y bridges.

Usa dos atributos:

stp; puede tomar los valores *"on"* / *"off"*, que habilita/deshabilita el protocolo STP(Spanning Tree Protocol) para los bridges creados por el parser.

promisc; puede tomar los valores *"on"* / *"off"*, que habilita/deshabilita el modo promiscuo para las interfaces configuradas por el parser de VNX.

<vm_mgmt>

Opcional. Única.

Con esta etiqueta se configuran varias especificaciones relacionadas con las interfaces de gestión de las maquinas virtuales. Es una forma de interactuar con las maquinas virtuales proporcionando una red de gestión entre el host y las maquinas virtuales, típicamente a través de SSH.

Esta etiqueta tiene los siguientes atributos:

type (obligatorio): Selecciona el tipo de interfaces de gestión, los valores permitidos son:

Private: Con este atributo se crea una interfaz de gestión por cada máquina virtual en el host llamada "name-e0", donde name es el nombre de la máquina virtual, usando un puente virtual. Esta interfaz del host se conecta con la máquina virtual usando una conexión punto a punto, con una máscara /30. En el lado de la máquina virtual, el nombre de la interfaz es 'eth0'. Las interfaces de gestión en ambos lados son configuradas dinámicamente con IPv4, y dependen de los atributos network, mask y offset. Para configurar las interfaces de gestión como private, el usuario debe tener privilegios de administración para poder configurar bridges virtuales.

Net: La interfaz de gestión de cada máquina virtual (llamada 'eth0') se conecta a una red virtual uml_switched. Las interfaces de gestión en las maquinas virtuales son configuradas dinámicamente con direcciones IPv4. Las direcciones ip son asignadas en dependencia de los atributos network, mask, offset.

None: No se configuran interfaces de gestión en ninguna máquina virtual. Esto es equivalente a poner: ***<mng_if>no</mng_if>*** para todas las maquinas virtuales.

network: es opcional y por defecto toma el valor 192.168.0.0.

mask: es opcional y por defecto toma el valor 24

offset: es opcional y por defecto toma el valor 0

Los atributos network, mask y offset especifican el rango de direcciones IPv4 para la red de gestión. Las asignaciones de direcciones comenzaran en la primera del rango de la red, más el valor del offset.

Un ejemplo sería:

<vm_mgmt type="..." network="10.0.0.0" mask="24" offset="4" />

Las direcciones comenzarían en 10.0.0.5 (la primera 10.0.0.1, más el offset 4). No se usarían ni la dirección de red 10.0.0.0, ni la de broadcast 10.0.0.255.

Las direcciones de gestión asignadas a cada máquina virtual dependerían del type escogido:

Si fuera private se establecerían conexiones punto a punto entre el host y cada máquina virtual quedando como muestra la figura 1.5:

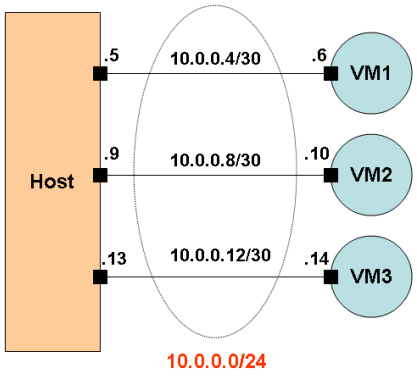


Figura 1.5. Red de gestión para atributo type = private.

En caso de utilizar el atributo type como net se asignan las direcciones secuencialmente comenzando con la primera más el offset, o sea 10.0.0.5, la segunda 10.0.0.6, y así sucesivamente. La dirección de gestión del host tendría que definirse en el atributo hostip de la etiqueta <mgmt_net>. Ver figura 1.6:

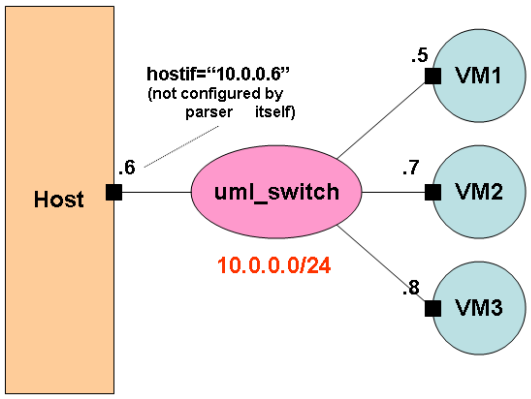


Figura 1.6. Red de gestión para atributo type = net.

<mgmt_net>

Esta etiqueta es obligatoria, cuando es usado el atributo type="net" de la etiqueta ***<vm_mgmt>***, de lo contrario está prohibido usarla. Define el comportamiento de la red de gestión uml_switched, al usar type="net" en ***<vm_mgmt>***.

Estos son los atributos de la etiqueta:

sock (obligatorio). El camino del controlador de socket de UNIX asociado con el proceso uml_switch que va a ser iniciado para la red de gestión.

hostip (obligatorio). La dirección IP de la interfaz que será asociada al proceso uml_switch para la red de gestión. Esta dirección tiene estar en el rango de la subnet establecida en la etiqueta ***<vm_mgmt>***.

autoconfigure (opcional). Permite la configuración automática del socket de gestión y de la red asociada en el host, así el usuario no tiene que preocuparse de los posibles errores del procedimiento. Este atributo solo se interpreta cuando se ejecuta VNX con el usuario root.

<host_mapping>

Opcional cuando el atributo type no es igual a "none", en la etiqueta ***<vm_mgmt>***, en caso contrario estaría prohibido.

Cuando esta etiqueta es usada los nombres de las maquinas virtuales (atributo name de la etiqueta ***<vm>***) son mapeadas a las dirección IPv4 asignada para la red de gestión de dicha máquina. El mapeo se realiza editando el archivo /etc/hosts. Por lo que se puede acceder a la máquina virtual a través de la dirección IP de gestión asignada o con su nombre mapeado en el archivo hosts. Solo los usuarios con privilegios como root pueden realizar cambios en el archivo hosts, por lo que para ejecutar esta etiqueta, se debe ejecutar el VNX como usuario root.

La etiqueta ***<host_mapping>*** es hija de ***<vm_mgmt>***.

<tun_device>

Opcional (valor por defecto: /dev/net/tun). Única.

Especifica donde reside el dispositivo tun usado para crear las interfaces de redes virtuales.

<vm_defaults>

Opcional. Única.

Esta etiqueta especifica valores por defecto a usar en todas las maquinas virtuales. Especificadas individualmente en cada ***<vm>.***

Como regla general la existencia de cualquier etiqueta dentro de <vm> anula a las etiquetas contenidas en <vm_defaults>.

Las etiquetas que podemos especificar dentro de <vm_defaults> son:

- <filesystem>
- <mem>
- <kernel>
- <shell>
- <basedir>
- <mng_if>
- <console>
- <route>
- <forwarding>
- <user>
- <filetree>
- <exec>

1.4.3 Redes Virtuales: *<net>*

La etiqueta <net> configura las redes virtuales del escenario de simulación. Cada red creada con <net> consiste en una interconexión virtual de un punto con las maquinas ya sean virtuales o el host. La configuración de direcciones IP y máscara no se realiza en la etiqueta <net>, sino en las etiquetas <if> y <hostif>.

Los atributos de esta etiqueta son:

- name (Obligatorio): que se encarga de identificar la red. Este nombre es usado por el atributo net de la etiqueta <if> y <hostif> para especificar la red a la que se conecta la interfaz de la máquina virtual o el host. El nombre "lo" es una palabra reservada y no se puede utilizar como nombre.

- mode (Obligatorio): Puede tomar dos valores:

a) "virtual_bridge": Es una red virtual basada en puentes virtuales.
b) "uml_switch": Es una red basada en UML switch.

- type (Opcional): Pueden implementarse dos tipos de redes:

a) "lan": Red LAN, esta es una configuración de una red convencional Ethernet.
b) "ppp": Esta red emula un enlace PPP entre dos maquinas virtuales. Las redes PPP tienen las siguientes características especiales:
    - Solo dos maquinas virtuales pueden conectarse a esta red PPP.
    - El atributo 'pointopoint' es usado en la configuración de la red de las

maquinas virtuales.

- Debe ser especificado un ancho de banda para el enlace usando <bw>.

- Para las redes basadas en puentes virtuales. Se utiliza el atributo **external** para la interconexión de una red virtual con redes externas a las que el host está conectado. El valor de este atributo será el nombre de la interfaz del host que está conectado a la red externa. Adicionalmente en caso de usar VLAN, será especificado un valor para el atributo **vlan** (número de VLAN).

- Para las redes uml_switch, pueden ser configurados los siguientes atributos:

  - **hub**: Al asignarle el valor "yes", se configura el proceso uml_switch en modo hub (el comportamiento por defecto es como switch).
  - **sock**: contiene el nombre del socket en el que se ejecuta un proceso uml_switch en el host. De esta forma el usuario root puede crear switches y conectarlos a algunas interfaces externas, permitiendo la comunicación con redes externas, a las cuales el host esté conectado.
  - Existen además atributos relacionados con la captura de tráfico, para su uso se necesita una versión especial de uml_switch_binary, por lo que quizás se necesitará hacer uso de la etiqueta **uml_switch_binary**:
    - El atributo **capture_file** define el archivo donde los paquetes capturados serán guardados.
    - **capture_expression** establece el filtro que será aplicado a la captura de tráfico. Estas expresiones utilizan la sintaxis de tcpdump.
    - El atributo **capture_dev** indica el nombre de la interfaz que puede ser usada con herramientas como ethereal para el estudio de paquetes de red en tiempo real.

<bw>

Obligatorio cuando el atributo type de la etiqueta ***<net>*** es "ppp", de lo contrario es prohibido. Especifica el ancho de banda que será usado en un enlace PPP.

1.4.4 Maquinas Virtuales. *<vm>*

Cada etiqueta ***<vm>*** describe una máquina virtual. Contiene atributos como:

- **name** (Obligatorio): especifica el nombre de la máquina virtual.

- **type** (Obligatorio): especifica el tipo de máquina virtual. Puede tomar los siguientes valores:
  - uml: define y ejecuta escenarios complejos basados en UML (User Mode Linux).
  - libvirt: utiliza libvirt para interactuar con las capacidades de

virtualización del host, permitiendo el uso de la mayoría de plataformas de virtualización disponibles para Linux.
        - dynamips: Permite la virtualización de routers Cisco.

- **subtype**: Prohibido en caso de que el type escogido sea uml, en caso contrario obligatorio y puede tomar los valores:
    - kvm: En caso de que el type escogido sea libvirt.
    - 3600 o 7200: en caso de que el type sea dymanips.

- **os**: Prohibido en el caso que el atributo type escogido sea uml o dymamips, de lo contrario es obligatorio y puede tener los siguientes valores (según el filesystem utilizado):
    - linux
    - freebsd
    - windows
    - olive
- **order** (Opcional): Establece el orden en que serán procesadas las maquinas virtuales. En caso de no especificarse se procesa según el orden en que este escrito el fichero VNX.
- **exec_mode** (Opcional): Define la forma en que se ejecutan comandos en toda la máquina virtual. Los valores que puede tomar dicha etiqueta, dependen del tipo de máquina virtual:

    - uml -> mconsole, net
    - libvirt-kvm-linux -> sdisk, cdrom, net
    - libvirt-kvm-freebsd -> sdisk, cdrom, net
    - libvirt-kvm-windows -> cdrom, net
    - libvirt-kvm-olive -> sdisk, net
    - dynamips -> telnet

  Cuando el atributo **exec_mode** toma el valor "**mconsole**", los comandos son ejecutados usando la consola uml, y un directorio compartido del filesystem del host (entre el host y la máquina virtual). En el caso de que tome el valor "**net**" los comandos son ejecutados usando la red de gestión, por lo que se debe configurar antes la red de gestión con la etiqueta ***<vm_mgmt>***.
  Para el caso de "**sdisk**" los comandos son ejecutados usando el disco local del host y la máquina virtual. Cuando **exec_mode** toma valor "**cdrom**" los comandos serían ejecutados entre el disco cdrom del host y la máquina virtual. Y finalmente en el caso de que la máquina virtual sea de tipo dynamips, puede tomar el valor "telnet", donde los comandos son ejecutados a través de una consola telnet.

<filesystem>

Esta etiqueta es opcional si se especifica por defecto un filesystem en ***<vm_defaults>***.

Cada máquina virtual necesita un filesystem, por lo que en esta etiqueta se especifica la ruta en la cual se encuentra. Utiliza el atributo type para especificar la forma de usar el filesystem y puede tomar los siguientes valores:

- type="direct"; indica que es el archivo que será usado para bootear la máquina virtual. Si se usa este modo directo el archivo debe tener permisos de escritura, para el usuario que ejecuta el vnx.

- type="cow"; indica que el filesystem será usado de manera copied-on-write (COW). Las modificaciones del archivo residirán en el directorio del parser de VNX. Ahorra gran cantidad de especio por lo que su uso es recomendado. Hay que tener en cuenta de que si se realiza una actualización del filesystem, es recomendable borrar las modificaciones salvadas, ya que podría ser no compatible.

- type="hostfs"; indica un directorio del hostfs en el que el usuario que ejecuta el parser de vnx tiene permisos de escritura y lectura.

La opción copy fue eliminada desde el DTD 1.7.

<mem>

Es opcional, los valores por defecto se especifican en ***<vm_defaults>***. Se especifica la cantidad de memoria que tendrá la máquina virtual. Los sufijos del valor pueden ser M para Megas y G para Gigas.

<kernel>

Obligatorio en caso de que el atributo type de ***<vm>*** sea "uml", en caso contrario no es necesario.

Específica la ruta del kernel UML para bootear la máquina virtual. Este archivo debe ser ejecutable. Esta etiqueta permite varios atributos, todos opcionales:

- initrd. Si el kernel es compilado con soporte initrd, entonces este atributo indica la localización del initrd apropiado, en casi contrario este atributo no será usado.
- root. Especifica explícitamente el dispositivo a usar por el root filesystem.
- devfs. Especifica si es un devfs que se pueda montar.
- modules. Especifica un directorio del filesystem del host que remplazará el directorio lib/modules en la máquina virtual que usa ese kernel.
- trace. Los valores permitidos son: "on", "off"; si es escogido "on", las trazas detalladas de inicio serán mostradas en un terminal, usualmente para la

detección de errores con el booteo de un UML, si se omite se asume el valor "off".

<shell>

Etiqueta opcional. Los valores por defecto se especifican en ***<vm_defaults>***.

Especifica el interprete Shell a usar para los scripts generados por el parser de VNX.

<basedir>

Establece la ruta raíz de la etiqueta **<filetree>**: el camino especificado en ***<filetree>*** continúa con el valor especificado en ***<basedir>***, siempre que el camino especificado en ***<filetree>*** no comience con "/". En caso de no definir ***<basedir>***, y ***<filetree>*** no comienza con "/", entonces VNX interpretará que la localización del archivo va a ser la misma donde se encuentra en documento XML. Esta etiqueta permite la fácil movilidad de archivos de configuración que serán cargados en las maquinas virtuales usando la etiqueta ***<filetree>***.

Un ejemplo sería:

<basedir>/tmp/confs</basedir>; esta sería la ruta base, luego para cada máquina virtual pondríamos una línea como esta:

<filetree root="/etc/program">host1</filetree>; aquí indicaríamos que los archivos se copiarían de la ruta tmp/confs/host1 y en la ruta /etc/program de la máquina virtual. En caso de que quisiéramos cambiar la ruta base, solo habría que modificar la etiqueta ***<basedir>***.

<mng_if>

Es opcional, los valores por defecto se pueden especificar en la etiqueta ***<vm_defaults>***. Si se indica: <mng_if>no</mng_if>, se está especificando que no se configurará una interfaz de gestión para esta máquina virtual. Cualquier otro valor como: <mng_if>yes</mng_if> o incluso <mng_if />, significa que si se habilita la interfaz de gestión.

<console>

Esta etiqueta define la forma en que se ejecutará la máquina virtual, utiliza atributos como **id** y **display**.

id puede tomar diferentes valores según el tipo de máquina virtual los cuales son:

- Para maquinas virtuales tipo libvirt:
  - **0**: consola gráfica
  - **1**: consola modo texto

- Para maquinas virtuales tipo dynamips:
    - **1**: consola principal
    - **2**: consola auxiliar (solo disponibles para routers C7200)

- Para maquinas Olive:
    - **1**: consola modo texto

display puede tomar los valores **yes** o **no**, los cuales indican si será ejecutado o no, según la **id** especificada, un ejemplo sería:

```
<vm_defaults>
        <console id= “0” display= ”no”/>
        <console id= ”1” display= “yes”/>
</vm_defaults>
```

<if>

Esta etiqueta describe las interfaces virtuales de la máquina virtual. Usa dos atributos id y net. El atributo id identifica la interfaz, o sea el nombre de la interfaz con una id=n, sería eth*n*. Si esta conectada a una red que tiene como atributo virtual_bridge, o la máquina virtual esta usando la red de gestión con atributo private, entonces la interfaz creada en el host se llamaría *name*-eth*n*, donde name es el nombre de la máquina virtual.

Los valores para el atributo id comienzan en 1, el 0 está reservado para la red de gestión.

El atributo net, específica la red virtual a la que se conecta a esa interfaz. La palabra “lo” está reservada para configurar las interfaces de loopback, o sea que en el caso de que el valor del atributo net sea “lo” entonces se configura la interfaz de loopback.

En el caso en que la máquina virtual sea del tipo dynamips, entonces se adiciona el atributo name a la etiqueta ***<if>***. Un ejemplo sería: name="e0/1".

Dentro de la etiqueta ***<if>*** pueden configurarse varias etiquetas como son:

<mac>

Específica la dirección MAC de la interfaz, en caso de no especificarse el parser de VNX la asigna automáticamente (si se utiliza la etiqueta ***<automac>***).

<ipv4>

Especifica la dirección IPv4 para la interfaz, la mascara puede ser especificada como parte del valor de la etiqueta, por ejemplo: <ipv4>10.1.1.1/24</ipv4>.

<ipv6>

Especifica la dirección IPv6 asignada a la interfaz, la mascara puede ser especificada como parte del valor de la etiqueta, por ejemplo: <ipv6>3ffe::3/64</ipv6>.

<route>

Etiqueta opcional, especifica una ruta estática que será configurada en la tabla de rutas de la máquina virtual, al iniciar el escenario VNX. Se usan dos atributos para esta etiqueta: **type**; que puede tomar dos valores "ipv4" para las rutas con direccionamiento IPv4 y "ipv6" para las rutas con direccionamiento IPv6. Y el segundo atributo **gw** que especifica la dirección del gateway para dicha ruta.

La red a alcanzar, a través de la ruta estática, por defecto para IPv4, es 0.0.0.0/0 y para IPv6 2000::/3

<forwarding>

Es opcional y los valores por defecto se pueden especificar en ***<vm_defaults>***. Esta etiqueta activa el reenvío de paquetes IP que llegan de una interfaz a otra de la máquina virtual, haciendo uso de su tabla de enrutamiento. Se utiliza el atributo opcional type (por defecto “ip”) y los valores permitidos son: "ipv4" que activa el reenvío de direcciones IPv4, "ipv6" que habilita el reenvío de direcciones IPv6 y finalmente "ip" que habilita ambas direcciones.

<user>

Opcional, permite la creación de usuarios en la máquina virtual al iniciar el escenario VNX, utiliza dos atributos:

- username: establece el nombre del usuario. En caso de que exista este usuario no se crea de nuevo, solo se adiciona a los nuevos grupos, de esta forma se mantiene como miembro en los grupos a los que ya pertenecía. Si se necesita eliminarlo de algún grupo habrá que realizar la acción manual desde la máquina virtual.

- group: establece el grupo al cual pertenecerá dicho usuario. En caso de no especificarse esta etiqueta, se utilizará el que tenga por defecto el sistema de la máquina virtual.

<group>

Es una etiqueta opcional que permite adicionar grupos a los cuales el usuario será adicionado.

<ssh_key>

Opcional, permite que la llave pública contenida en cada etiqueta ***<ssh_key>***, y que a su vez está dentro de la etiqueta ***<user>***, será ubicada en el archivo $HOME/.ssh/authorized_keys, donde $HOME es el directorio home del usuario. En caso de ya existir no volverá a ser añadida.

<filetree>

Etiqueta opcional, específica un directorio de archivos en el sistema de archivos del host, que serán copiados al sistema de archivos de la máquina virtual (sobrescribirá los archivos existentes). Esta etiqueta permite la fácil copia de directorios de archivos de configuración que serán utilizados en la simulación.

En caso de que la ruta del directorio comience con la raíz "/", entonces se está especificando la ruta absoluta. En caso de no comenzar por "/", entonces continuará detrás del valor establecido en la etiqueta ***<basedir>***, sino se definiera ***<basedir>***, la ruta sería donde está salvado el archivo XML.

Tres atributos son usados en esta etiqueta:

root: Especifica en que lugar del sistema de archivos de la máquina virtual será copiado el filetree.

seq: Indica el nombre de la secuencia de comandos que activan la copia. La copia del árbol de archivos ser realiza antes de la ejecución de comandos ***<exec>***.

user: Opcional, especifica el usuario para la realización de la copia. Si no se especificara se usaría por defecto el usuario root.

<exec>

Esta etiqueta es opcional y se encarga de la especificación de un comando para ser ejecutado por la máquina virtual durante el modo de ejecución de secuencias de comandos. Los atributos son:

seq: Obligatorio. Es una cadena que define la secuencia de comandos. Esta se utiliza con el parámetro –x: un ejemplo sería; *sudo vnx -f archivo.xml -v -x cadena*. El orden de ejecución de los comandos va a ser el mismo en el que aparecen en el archivo VNX.

type: Obligatorio. Los valores permitidos son "verbatim" y "file". Al usar "verbatim" se está especificando que la reproducción exacta del valor de la etiqueta es el comando a ejecutar, mientras que al usar "file" el valor de la etiqueta será la ruta que apunta a un

archivo en el sistema de archivos del host, que contiene una secuencia de comandos que serán ejecutados línea por línea.

ostype: Opcional. Este atributo se utiliza para especificar el tipo de comando a incluir en la etiqueta ***<exec>***, además de si será en modo texto (CLI) o gráfico (GUI) y si la herramienta VNX esperará que la ejecución termine o no:

ostype	**Tipo de comando**	**Espera por que termine la ejecución**
system	CLI	YES
exec	CLI	NO
xsystem	GUI	YES
xexec	GUI	NO

Tomarán los siguientes valores en dependencia de tipo de máquina virtual que sea:

uml -> system

libvirt-kvm-linux -> system; exec; xsystem; xexec

libvirt-kvm-freebsd -> system; exec; xsystem; xexec

libvirt-kvm-windows -> cmd; system; exec

libvirt-kvm-olive -> show; set; load; system

dynamips -> show; set; load

user: Opcional, especifica el usuario para la ejecución de los comandos. Si no se especificara se usaría por defecto el usuario root. Este atributo no esta permitido dentro de la etiqueta ***<host>***, ya que en el host los comandos deben ser ejecutados por el root.

1.4.5 Configuración HOST *<host>*

Esta etiqueta es opcional y especifica la configuración para el host. Es muy similar a la etiqueta para las maquinas virtuales ***<vm>***. Particularmente las etiquetas ***<route>***, ***<forwarding>*** y ***<exec>*** son usadas con la misma sintaxis descrita para las máquinas virtuales. Adicionalmente existen otras etiquetas que solo pueden ser usadas dentro de ***<host>***:

<hostif>

Esta etiqueta es muy similar al ***<if>*** de las maquinas virtuales, pero con las siguientes diferencias:

- El atributo net es usado con la misma sintaxis, pero el nombre de la interfaz en el host, va a ser el mismo que el valor del atributo. O sea que si net=NET0, el nombre de la interfaz en el host sería NET0. Y además el atributo id no es usado.

- La etiqueta ***<mac>*** no es usada.

<physicalif>

Opcional. Esta etiqueta es usada por el parser de VNX solamente cuando la topología es destruida, o sea que cuando se destruye las redes virtuales definidas en la etiqueta ***<net>***, la interfaz física no recupera sus valores anteriores automáticamente. Para su correcta recuperación automática tiene que definirse la etiqueta ***<physicalif>***. Los atributos utilizados serían **type** con los valores "ipv4" o "ipv6", **name** para el nombre de la interfaz, **ip** con el valor de la dirección IP (para IPv4 solo la dirección IP y para IPv6 incluida la máscara, ejemplo /64), **mask** para la mascara con IPv4, ya que con IPv6 no tendría sentido y **gw** con la puerta de enlace por defecto.

Como mucho dos ***<physicalif>*** pueden ser definidas por interfaces: una type = "ipv4" y otra type = "ipv6".

Capítulo 2: GTK-Perl

Al ser el objetivo principal del trabajo, la creación de una interfaz gráfica para VNX, mediante los programas Perl (lenguaje de programación) y Glade (diseño de interfaces gráficas), este capítulo se centrará en una descripción teórica del GTK en combinación con Perl.

GTK (GIMP toolkit) es una librería utilizada para la creación de interfaces gráficas, o sea, es un conjunto de herramientas orientadas a objetos, de plataforma cruzada y lenguaje neutral, que es ante todo, usada para crear aplicaciones independientes de GNOME. Fue originalmente creado para el programa GIMP (GNU Image Manipulation Program). Es un software de código abierto. Las librerías de GTK están escritas en el lenguaje de programación C, pero pueden ser usadas por varios lenguajes como Perl, Python, Ruby, PHP, Java y otros (8).

Otro elemento importante de GTK, es que está constituido por colecciones de widgets, los cuales no son más que controles GUI como botones, menús, cajas de diálogo y otros objetos relacionados a funciones generales.

El vínculo existente entre Perl y GTK, es que GTK-Perl es un modulo que permite tener acceso a las librerías GTK a través del lenguaje Perl. Está disponible bajo la licencia GPL, lo que significa que posee las mismas restricciones y las mismas libertades que Perl.

GTK-Perl incluye varios módulos (GdkImlib, Gnome, GtkXmHTML, GtkGLArea y Glade).

2.1 Widget

Pese a estar desarrollado en el lenguaje C, GTK cuenta con una jerarquía de objetos para organizar los controles y elementos que forman parte de él. El sistema de objetos que se utiliza es GObject, un potente componente de la arquitectura de desarrollo de GNOME. Todas las clases en dicho sistema utilizan como clase base GObject.

GtkWidget es la clase base para todos los controles que están disponibles en GTK. Toda ventana, botón, caja de texto, o menú emergente que sea parte de una aplicación, será una subclase de GtkWidget, que a la vez es una subclase de GObject (9).

Creación de un widget

Los pasos principales para crear un widget en Gtk-Perl son:

1. Nueva función GTK::Widget() para la creación de un nuevo widget. Los widgets disponibles serán detallados mas adelante en este capítulo.

2. Conectar todas las señales y eventos que se quieren usar a los manejadores apropiados.

3. Establecer los atributos del widget.

4. Ubicar el widget en un contenedor usando la llamada apropiada como: $container->add() o $box->pack_start().

5. Mostrar el widget; show().

Finalmente la función show() le permite conocer a Gtk-Perl que han sido configurados todos los atributos del widget y que está listo para ser visualizado. También es posible ocultar el widget con la función hide(). El hijo de un widget no será mostrado hasta que no haya sido mostrado el widget padre con la función show(). Recordar también que una ventana es un widget también.

Jerarquía de Widget

Los widgets de GTK poseen una estructura de clases jerárquica. Todo elemento hereda de GtkWidget y este a su vez hereda de GtkObject. Cada hijo agrega funcionalidad y estado a su padre. Cada nueva rama en la jerarquía muestra una separación de comportamiento con respecto a sus ramas hermanas.

En el anexo 1 se muestra el árbol de jerarquía de clases utilizada para la implementación de los widgets. El conocimiento de esta jerarquía es muy útil porque nos dice qué funciones están disponibles para cada widget.

Creación y destrucción de un widget

Para la creación de una instancia de un widget la función utilizada es:

```
$widget = new Gtk::Widget();
```

Para la destrucción del widget se usaría la siguiente función:

```
$widget->destroy();
```

El widget debe tener una función de devolución de llamada (callback) registrada para la señal destroy. Con lo que al llamar a esta función callback, el widget sería destruido.

Mostrar y ocultar un widget

Las siguientes funciones se encargan de mostrar y ocultar un widget:

```
$widget->show();

$widget->show_now();

$widget->show_all();

$widget->hide();

$widget->hide_all();
```

Sensibilidad widget

Un widget insensible es aquel que no responde a ninguna entrada, o pudiera decirse también como la falta de inactividad del widget. Un widget puede ser configurado como sensible utilizando la siguiente función:

```
$widget->set_sensitive( $sensitive );
```

El argumento ($sensitive) es TRUE (verdadero) o FALSE (falso), que establece si el widget será sensible o no. Sin embargo aun declarando la sensibilidad de un widget, si el widget padre es insensible, el hijo también lo sería. En caso de querer testear si el widget es sensible se utiliza la siguiente función:

```
$widget->sensitive();
```

Otra forma de testear sensibilidad es con la siguiente función:

$widget->is_sensitive();

Esta difiere de la anterior en que además de testear si el widget es sensible, testea al widget padre y así sucesivamente. O sea que solo con esta función es posible decir si un widget es verdaderamente sensible.

Establecer posición y tamaño de un widget

Es posible establecer la posición y tamaño de un widget con las siguientes funciones:

$widget->set_uposition($x, $y);

$widget->set_usize($width, $height);

La función set_uposition() establecerá la posición del widget a $x pixeles desde la izquierda y $y pixeles desde arriba. Y la función set_usize() establecerá el tamaño del ancho del widget, al número de pixeles definidos en $width y la altura al número establecido en $height.

2.2 Contenedores

Los contenedores son widgets que contienen otros widgets, estos tienen la jerarquía de herencia mostrada a continuación:

```
Object
+--- Widget
+--- Container
+--- Box
```

El primer tipo de contenedor es una subclase de bin, que a su vez es hijo de un contenedor. Este tipo de contenedores solo puede almacenar un solo hijo y es usado para darles funcionalidad. Un buen ejemplo de este tipo de contenedores son los botones y marcos (frames).

El segundo tipo de contenedor puede almacenar múltiples widgets y son usados para administrar su distribución. Ejemplos de este tipo de contenedores son los las cajas y tablas.

Adición y eliminado de widgets

Los widgets pueden ser adicionados o eliminados de un contenedor usando las siguientes funciones:

$container->add($widget);

$container->remove($widget);

Una vez adicionado a un contenedor, el widget no será visible hasta que ambos llamen a la función show(), o hasta que el contenedor llame a la función show_all().

Obtención de contenedores hijos

La siguiente función devuelve una lista de los widgets que se encuentran dentro de un contenedor:

@children = $container->children();

Configuración del ancho de borde de un contenedor

El ancho de borde es el número de pixeles entre el widget hijo y el borde del contenedor. Esto puede ser configurado utilizando las siguientes funciones:

$container->border_width();

$container->set_border_width();

La primera función es definida en GTK como una macro de la segunda, por lo que no existe diferencia entre las dos, aunque normalmente siempre se utiliza más la segunda función.

2.3 Señales y eventos

GTK es un conjunto de herramientas orientada a eventos, lo que significa que el main de GTK está en espera, hasta ocurrir un evento y el control es pasado a la función apropiada.

Este paso de control se realiza usando la idea de señales, o sea cuando ocurre un evento como presionar un botón del ratón, se emite una señal por parte del widget que es presionado. Así es como GTK realiza la mayor parte de su trabajo. Existen señales que todos los widgets heredan, como es el caso de: “destroy”, y existen señales que son especificas de cada widget como es el caso de la señal “toggled” en un botón de activación.

Para hacer que un widget realice una acción se crea un manejador de señales para captar las señales y llamar a la función apropiada. Esto se hace usando una función como estas:

```
$object->signal_connect( "signal_name", \&signal_func );

$object->signal_connect( signal_name => \&signal_func );

$object->signal_connect( "signal_name", \&signal_func, $optional_data ... );

$object->signal_connect( "signal_name", \&signal_func, @optional_data );
```

Para Perl las dos primeras formas son idénticas y las dos ultimas también son idénticas, ya que Perl envía todos los argumentos como una lista simple de escalares. Por su puesto se puede enviar tantos elementos en la lista como se desee.

La variable de la izquierda en la función de signal_connect() es el widget que emite la señal. El primer argumento es una cadena que representa la señal que se quiere registrar con la devolución de llamada. El segundo argumento es la referencia a la subrutina a la que se quiere llamar. Esta subrutina es llamada devolución de llamada (callback). Si se quiere pasar datos a la función callback, se puede especificar poniéndolo al final de la lista de argumentos. Todo después del segundo argumento, es pasado como una lista, por lo que es posible pasar tantos argumentos a la función callback como se desee.

Para las funciones callback que son asociadas con una sola señal, y tienen pocas líneas, es común ver algo como esto:

```
$object->signal_connect( "signal_name", sub { do_something; });
```

Una función callback usualmente es definida de la siguiente manera:

```
sub callback_func
{
my ( $widget, @data ) = @_;
...
}
```

El primer argumento enviado a la función callback siempre será el widget que emite la señal y el resto de los argumentos son datos opcionales enviados por la función signal_connect().

Eventos

Además del mecanismo de señales descrito anteriormente, existen una serie de eventos que reflejan el mecanismo de eventos X. La función callback también puede ser conectada a estos eventos. Estos eventos son:

- event
- button_press_event
- button_release_event
- motion_notify_event
- delete_event
- destroy_event
- expose_event
- key_press_event
- key_release_event
- enter_notify_event
- leave_notify_event
- configure_event
- focus_in_event
- focus_out_event
- map_event
- unmap_event
- property_notify_event
- selection_clear_event
- selection_request_event
- selection_notify_event
- proximity_in_event
- proximity_out_event
- drag_begin_event
- drag_request_event
- drag_end_event
- drop_enter_event
- drop_leave_event
- drop_data_available_event
- other_event

Para conectar una función callback con uno de estos eventos, se puede usar la función signal_connect(), del mismo modo en que se conecta una señal, utilizando sólo uno de los nombres de eventos anteriores en lugar de un nombre de señal. El último argumento pasado a la función callback es la de estructura de eventos, por lo que al principio de la función se debe poner así:

```
my ( $widget, $data, $event ) = @_;
```

En caso de que se quiera pasar un arreglo de datos:

```
my ( $widget, @data ) = @_;

my $event = pop( @data );
```

Algunos de los campos más comunes en la estructura de eventos son button, keyval, y type.

El campo button contiene un número con el botón pulsado (típicamente 1, 2 y 3). El campo keyval contiene la tecla que fue presionada. Y el campo type contiene una de las siguientes cadenas:

- ’nothing’
- ’delete’
- ’destroy’
- ’expose’
- ’motion_notify’
- ’button_press’
- ’2button_press’
- ’3button_press’
- ’button_release’
- ’key_press’
- ’key_release’
- ’enter_notify’
- ’leave_notify’
- ’focus_change’
- ’configure’
- ’map’
- ’unmap’
- ’property_notify’
- ’selection_clear’
- ’selection_request’
- ’selection_notify’
- ’proximity_in’
- ’proximity_out’
- ’drag_begin’
- ’drag_request’
- ’drop_enter’
- ’drop_leave’
- ’drop_data_avail’
- ’cLIent_event’
- ’visibiLIty_notify’
- ’no_expose’

Estos campos hacen más fácil la determinación de la causa de un evento.

Valor de retorno de una función callback

El tipo de retorno de la función signal_connect() es una etiqueta que identifica la función callback. Se pueden tener cuantas funciones callback por señal y por objeto como se necesite, y cada una se ejecutará en el orden en que haya sido escrita.

Emisión de señales

Si se quiere emitir una señal específica puede emplearse una de las siguientes funciones:

$widget->signal_emit($id);

$widget->signal_emit_by_name($signal_name);

El argumento para la primera función es la etiqueta que retorna la función signal_connect(). Y el argumento de la segunda es la cadena que identifica el nombre de la señal.

Además, muchos widgets tienen funciones que emiten las señales más comunes. Por ejemplo la función destroy() causará la emisión de la señal "destroy", y la función activate() causará la emisión de la señal "activate". Estas funciones tienen preferencia sobre las funciones signal_emit_by_name().

Eliminar función callback

La etiqueta id que es devuelta por la función signal_connect() permite también eliminar una función callback de la lista de señales de un widget, usando la siguiente función:

$widget->signal_disconnect($id);

Además si se quisiera eliminar todas las señales de la lista de un widget, se puede usar esta función:

$widget->signal_handlers_destroy();

Esta función no es usada muy frecuentemente ya que es simple remover todas las señales de un objeto. Esto se hace automáticamente cuando el widget es destruido.

Deshabilitar temporalmente una función callback

Es posible deshabilitar los manejadores de señales con el siguiente grupo de funciones:

$widget->signal_handler_block($callback_id);

$widget->signal_handler_block_by_func(\&callback, $data);

$widget->signal_handler_block_by_data($data);

$widget->signal_handler_unblock($callback_id);

$widget->signal_handler_unblock_by_func(\&callback, $data);

$widget->signal_handler_unblock_by_data($data);

2.4 Contenedores básicos

En este epígrafe se realizará una breve descripción de los widgets contenedores más comunes de GTK.

Al crear una aplicación normalmente se quiere poner más de un widget dentro de una ventana. Para un solo widget resulta muy fácil utilizando la función $window->add(). Pero cuando se quiere poner más de un widget no se puede controlar su ubicación y es cuando son usadas las cajas de empaquetado (packing boxes).

El empaquetado se realiza creando cajas y almacenando widgets en su interior. Las cajas de empaquetado no son más que contenedores de widgets invisibles. Hay dos tipos de cajas de empaquetado las cajas horizontales y cajas verticales. Cuando se empaqueta widgets en una caja horizontal, los objetos son insertados horizontalmente de izquierda a derecha o viceversa. Y en las cajas verticales de arriba hacia abajo o viceversa. Además es posible usar cualquier combinación de ambas cajas para lograr el efecto deseado.

Para crear una caja horizontal se usa esta función:

$hbox = new Gtk::HBox($homogeneous, $spacing);

Y una caja vertical con la siguiente:

$vbox = new Gtk::VBox($homogeneous, $spacing);

El argumento homogeneous controla si cada objeto de la caja tiene el mismo tamaño y spacing es un espacio que se añade entre objetos.

Para introducir objetos dentro de una caja se utilizan estas funciones:

$box->pack_start($child, $expand, $fill, $padding);

$box->pack_end($child, $expand, $fill, $padding);

La función pack_start() comienza de arriba hacia abajo en una caja vertical, y de izquierda a derecha en una vertical y pack_end() realiza la operación contraria.

El primer argumento $child va a ser el widget a ser empaquetado en la caja, $expand controla si los widgets son expandidos en la caja para rellenar todo el espacio de la misma (TRUE), o si por el contrario no se usa el espacio extra dentro de la caja (FALSE). El argumento $fill controla si el espacio extra se mete dentro de los objetos (TRUE) o como relleno extra (FALSE). Sólo tiene efecto si el argumento de expansión

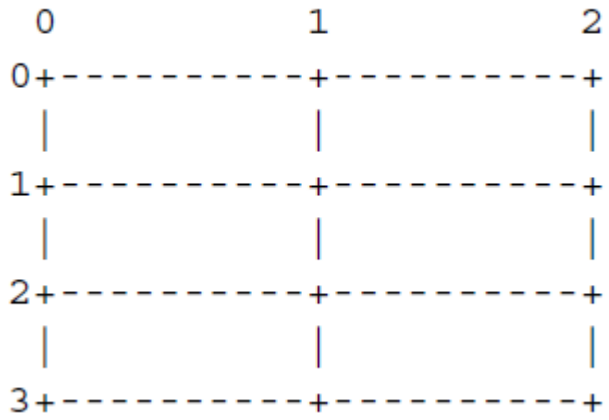
también es TRUE y finalmente $padding es un espacio que se añade a cada lado de un objeto.

También hay otro sistema de empaquetado: Tablas. Estas pueden ser extremadamente útiles en ciertas situaciones. Usando tablas, se crea una rejilla en la que colocar widgets. Los widgets pueden ocupar tantos espacios como se especifique.

La función que permite la creación de una tabla es:

$table = new Gtk::Table($num_rows, $num_columns, $homogeneous);

Como es lógico el primer argumento es el número de filas y el segundo el de columnas. El tercero establece el tamaño de las celdas de la tabla. Si es TRUE se fuerza a que el tamaño de las celdas sea igual al de la celda mayor. Con FALSE se establece el ancho de toda una columna igual al de la celda más ancha de esa columna, y la altura de una fila será la de la celda más alta de esa fila. El número de filas y columnas varía entre 0 y n, donde n es el número especificado en la función gtk_table_new. Así si se especifica columnas = 2 y filas = 3 la apariencia será parecida a:



Para adicionar un widget a una caja de una tabla se usa esta función:

$table->attach($child, $left_attach, $right_attach,

$top_attach, $bottom_attach, $xoptions,

$yoptions, $xpadding, $ypadding);

GTK:Window es la ventana de nivel superior que aparece en la ventana raíz e interactúa o no con el gestor de ventanas. Puede ser la ventana principal de una aplicación, un diálogo, o una ventana temporal.

Es posible utilizar múltiples funciones con GTK:Window, como es; establecer el titulo de una ventana a través de esta función: $window->set_title($title); o establecer el

tamaño por defecto de una ventana: $window->set_default_size($width, $height); o la posición: $window->set_position($position), etc.

2.5 Widget Botón

Existen dos formas de crear un widget botón, se puede crear un botón vacío y luego adicionar un widget hijo, o crear directamente el botón con su etiqueta. La segunda es la más recomendada. Estas funciones se encargan de crear un botón:

$button = new Gtk::Button();

$button = new Gtk::Button($label);

$button = new_with_label Gtk::Button($label);

La primera función crea un botón vacío y las dos últimas crean un botón con su etiqueta, la segunda función es en realidad una forma rápida de la segunda. Si se crea un botón con su etiqueta, es posible acceder al widget hijo (etiqueta), por ejemplo a través de esta función es posible acceder y cambiar la etiqueta:

$button->child->set("new label");

En caso de no crear directamente un botón con su etiqueta también es posible hacerlo por pasos como muestran estas funciones:

$button = new Gtk::Button();

$label = new Gtk::Label("text");

$button->add($label);

Los widgets botones tienen además un grupo de señales predeterminadas que resultan muy útiles a la hora de realizar una aplicación que incorpore botones:

- “pressed”; se emite cuando el botón del puntero se presiona en el control botón o cuando es llamada la función $button->pressed().
- “released”; se emite cuando el botón del puntero se suelta en el control botón o cuando es llamada la función $button->released().
- “clicked”; se emite cuando el botón del puntero se presiona y luego se suelta sobre el control botón, o cuando es llamada la función $button->clicked().
- “enter”; se emite cuando el puntero entra en el control botón o cuando es llamada la función $button->enter().

- “leave”; se emite cuando el puntero sale del control botón o cuando es llamada la función $button->leave().

Existen otros tipos de botones como son:

- Botones Biestado (Toggle Buttons)
- Botones de Activación (Check Buttons)
- Botones de Exclusión Mútua (Radio Buttons)

Los Botones Biestado derivan de los botones normales y son muy similares, excepto que siempre están en uno de dos estados, alternándolos con un clic. Puedan estar presionados, y cuando se vuelva a hacer clic, volverán a su estado inicial, levantados. Se hace clic otra vez y volverán a estar presionados.

Los botones Biestado son la base para los botones de activación y los botones de exclusión mútua, y por ello, muchas de las llamadas usadas con los botones biestado son heredados por los botones de activación y los botones de exclusión mútua.

Los botones de activación heredan muchas propiedades y métodos de los botones biestado vistos anteriormente, pero su apariencia es un poco diferente. En vez de ser botones con texto dentro de ellos, son pequeñas cajas con un texto a su derecha. Normalmente se utilizan para opciones que pueden estar activadas o desactivadas en las aplicaciones.

Los botones de exclusión mútua son similares a los botones de activación excepto que se agrupan, de tal forma que sólo uno puede estar seleccionado/pulsado en un momento dado. Esto es bueno para aquellas situaciones en las que la aplicación necesita seleccionar un valor entre una pequeña lista de opciones.

2.6 Widgets básicos

Hasta ahora se ha visto que GTK posee una amplia gama de widgets, que a su vez pueden contener más widgets en su interior, pero también existen widgets no son contenedores, desde los más simples como una etiqueta, hasta widgets más sofisticados como una barra de progreso.

A continuación se nombrarán algunos de ellos:

Etiquetas: Son usadas con mucha frecuencia en GTK, y son relativamente fáciles, las etiquetas no emiten ninguna señal y por lo tanto no son asociadas a una ventana gráfica. Si se necesitara que captara señales habría que ubicarla dentro de un botón o una caja de eventos. Para crear una etiqueta es usada esta función:

```
$label = new Gtk::Label( $string );
```

Separadores: Son los widgets más simples, son sólo las líneas con una sombra para hacer que parezca hundido en el fondo. Existen dos tipos de separadores vertical y horizontal y son creados con estas funciones:

$hseparator = new Gtk::HSeparator();

$vseparator = new Gtk::VSeparator();

Widget de información rápida (tooltip): Estos widgets son las pequeñas etiquetas que texto que aparecen cuando se sitúa el puntero del ratón sobre un botón u otro widget durante algunos segundos.

Widgets de rango: Estos incluyen el widget scrollbar y el widget de escala. Todos ellos contienen un canal y un control deslizante. Al arrastrar el deslizador con el puntero se mueve adelante y atrás dentro del canal, mientras que haciendo clic en el canal avanzas el control deslizante hasta la ubicación del clic. Todos los widgets de rango están asociados con un objeto de ajuste, con el cual se calcula la longitud del control deslizante y su posición dentro del canal.

Barra de progreso: Es usada para mostrar el estado de una operación. Existen dos formas de crear una barra de progreso, una muy simple que no toma argumentos, y la otra que toma un objeto de ajuste como argumento. En el primer caso la barra de progreso crea su propio objeto de ajuste:

$progress = new Gtk::ProgressBar();

$progress = new Gtk::ProgressBar($adjustment);

El segundo método tiene la ventaja que puede usar el objeto de ajuste para especificar sus propios parámetros.

Cuadros de diálogo: El widget de cuadro de diálogo es bastante simple, sólo es una ventana con algunas cosas ya preempaquetadas. Simplemente se crea una ventana en la cual se empaqueta una vbox, un separador y una hbox llamada``action_area''. Este tipo de widgets puede ser usado como mensajes *pop-up* (pequeñas ventanas con texto en su interior que aparecen cuando el usuario hace algo y queremos informarle de alguna cosa) y otras cosas parecidas.

Barras de estado: Son widgets usados para mostrar un mensaje. Todo aquello que haya sido mostrado se guarda en una pila, con lo que es muy fácil repetir el último mensaje. Para permitir que diferentes partes del programa usen la misma barra de estado éstas usan Identificadores por Contexto (Context Identifiers) para identificar a los `usuarios'. El mensaje que está en lo alto de la pila será el siguiente en mostrarse, sin importar el contexto en el que se esté. Los mensajes se almacenan en el orden el último en entrar es el primero en salir, y el Identificador por Contexto no influye en este orden.

Entrada de texto: El widget Entry permite mostrar e introducir texto en una línea de un cuadro de diálogo. El texto se puede poner con llamadas a funciones que permiten remplazar, preañadir o añadir el texto al contenido actual del widget Entry.

Selección de ficheros: El widget de selección de ficheros nos proporciona una forma rápida y sencilla de mostrar un cuadro de diálogo para la selección de un fichero. Ya viene con los botones Aceptar, Cancelar y ayuda. Una magnifica ayuda para acortar el tiempo de programación.

Ajustes: Existen diferentes widgets en GTK que pueden ser ajustados visualmente por el usuario mediante el ratón o el teclado. Ejemplo de esto son los widgets de selección. También hay otros widgets que pueden ser ajustados parcialmente, por ejemplo el widget de texto o el viewport. Como es lógico el programa tiene que poder reaccionar a los cambios que el usuario realiza. Mediante señales especiales se podría conseguir saber qué y cuánto ha sido ajustado, pero en el caso de que se quiera conectar la señal con diferentes widgets, de manera que todos cambien al mismo tiempo, el proceso puede ser un poco tedioso. El ejemplo más obvio es conectar una barra de desplazamiento a una región con texto. Si cada widget posee su propia forma de establecer u obtener sus valores de ajuste el programador puede que tenga que escribir sus propios controladores de señales para traducir el resultado de la señal producida por un widget como el argumento de una función usada para determinar valores en otro widget. Para resolver este problema GTK usa objetos del tipo GtkAdjustment. Con ellos se consigue almacenar y traspasar información de una forma abstracta y flexible. El uso más obvio es el de almacenes de párametros para widgets de escala (barras deslizantes y escalas). Como los GtkAdjustment derivan de GtkObject poseen cualidades intrínsecas que les permiten ser algo más que simples estructuras de datos. Lo más importante es que pueden emitir señales que a su vez pueden ser usadas tanto para reaccionar frente al cambio de datos introducidos por el usuario como para transferir los nuevos valores de forma transparente entre widgets ajustables.

2.7 Glade

Como se ha visto anteriormente existen numerosas funciones que nos permiten crear diferentes widgets para la creación de una aplicación gráfica con GTK. En este epígrafe se hará una descripción del programa Glade, utilizado más adelante para la creación de la interfaz gráfica de VNX, con el cual resulta mucho más cómodo, para el diseño de una interfaz gráfica en GTK.

Glade es una herramienta para un rápido desarrollo de aplicaciones GTK. Es una propia aplicación GTK. O sea es un software desarrollado para simplificar el proceso de diseño de la interfaz gráfica de la aplicación. Este programa crea un archivo glade, que no es más que un archivo XML que describe jerárquicamente los widgets que comprenden la interfaz (10).

Después de diseñar una interfaz gráfica con glade, el diseño y configuración es salvado en un fichero XML. Existe además la librería libglade que conoce como construir y conectar la interfaz gráfica diseñada. Lo que en conjunto permite a libglade junto con Perl crear y manipular aplicaciones graficas GTK. Resultando menos tedioso que cargar en un fichero Perl todas las funciones para la creación de la aplicación GTK.

2.7.1 Construcción de una interfaz gráfica

Al iniciar un proyecto nuevo de glade, se inician dos ventanas, la ventana principal, que es usada para crear la interfaz gráfica y el diálogo de preferencias que permite configurar algunos parámetros del proyecto. Ver anexo 2.

Como se observa en el anexo 2 la ventana principal está dividida en varias partes, a la izquierda está la paleta de widgets, que contiene los elementos de la interfaz que se va a crear y está estructurada según las diferentes clasificaciones de los widgets. En el centro se encuentra el área donde se irán adicionando los widgets para la creación de la aplicación GTK, y a la derecha tenemos el buscador de widgets y más abajo el editor de propiedades, esta ventana se activa cuando un widget está seleccionado y permite modificar sus atributos, así como establecer las señales que serán definidas.

Normalmente para el diseño de una interfaz debemos comenzar por los widgets de niveles superiores ya que se va a necesitar un lugar para añadir los widgets, para ello se da clic debajo de niveles superiores en el icono de ventana. Ver anexo3.

Un widget tipo ventana aparecerá en el espacio de trabajo preparado para ser modificado. En las propiedades de la ventana (bajo la pestaña General) se puede asignar un nombre más descriptivo para ser mostrado en la barra de título de la ventana en tiempo de ejecución, así como su tamaño predeterminado, podemos observarlo en el anexo 4.

Una vez preparada la ventana, debemos empaquetar widgets para permitir cualidades como el resizing (cambio de tamaño de la ventana) automático. Cuando un usuario cambia el tamaño de una aplicación, normalmente se quiere que los widgets de la ventana cambien también su tamaño para aprovechar mejor el nuevo espacio, ya sea expandiéndose o comprimiéndose. El empaquetado permite que esto se haga automáticamente y afortunadamente libera al programador de escribir código de adaptación al nuevo tamaño. El empaquetado se realiza creando cajas o tablas. Estos widgets son invisibles, lo que quiere decir que no se ven en el momento de la ejecución del programa, sin embargo tienen efecto en la aplicación.

Para realizar el empaquetado podemos pulsar el icono Caja Vertical en la paleta y después pulsar en cualquier lugar de la ventana que acabamos de crear. Si se indica que se desean 3 filas para la caja vertical aparecería algo como el anexo 5. La ventana queda ahora dividida en tres filas, preparada para añadir más widgets.

En cada fila es posible adicionar un widget, un ejemplo sería si en la primera fila adicionamos una etiqueta, luego en la segunda un widget entrada de texto, quedaría algo como el anexo 6.

O sea que por cada fila de la caja vertical podemos añadir un widget, pero si se quisiera en la tercera fila añadir dos botones, se tendría que añadir una caja horizontal de dos columnas y finalmente quedaría una pequeña interfaz gráfica como el anexo 7.

El ejemplo anterior es una interfaz gráfica muy sencilla, pero con ello se muestra el proceso básico para la creación de interfaces más complejas, para la cual además podría utilizarse una amplia variedad de widgets disponibles en Glade. Ver anexo 8.

Capítulo 3: Desarrollo de una interfaz gráfica para VNX

Este capítulo está dedicado al desarrollo de una interfaz gráfica para la herramienta VNX. Para ello se empleará parte del código utilizado en la interfaz gráfica VNUMLGUI, desarrollada para la anterior versión de VNX; VNUML.

VNUMLGUI es una interfaz gráfica desarrollada en Perl/GTK2. Y con ella es posible diseñar una topología de red, colocando gráficamente los diferentes elementos de red, como redes, maquinas virtuales, host y enlazándolos entre ellos. Una vez realizado el diseño gráfico, el programa genera automáticamente el fichero XML, listo para ser ejecutado por la herramienta de virtualización VNUML. Ver anexos 9 y 10.

Inicialmente se comenzará este capitulo realizando un estudio de cuales son las diferencias entre la herramienta VNX y VNUML, para posteriormente implementar los cambios en la interfaz gráfica y adaptarla a VNX, creando así la nueva versión de la interfaz gráfica para VNX, a la cual se le llamará VNXGUI.

3.1 Diferencias entre VNUML y VNX

La idea fundamental del programa gráfico para estas herramientas de virtualización es que una vez creada la topología gráfica de un escenario de red, diseñado las diferentes conexiones entre cada elemento de red, y configurado sus diferentes parámetros, genere un fichero XML, que contenga la estructura y sintaxis, que entiendan las herramientas de virtualización.

Por lo que este epígrafe se centrará en cuales son las modificaciones en el lenguaje del fichero XML de VNUML, en su nueva versión VNX. A continuación se irán mostrando:

1- Normalmente un fichero VNUML comenzaba con las siguientes líneas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
```

En la segunda línea era especificada la ruta del fichero DTD, especificando la correcta estructura de los elementos del documento XML, para VNX, esto cambia ahora es realizado con un esquema XSD, por lo que al iniciar un archivo VNX siempre comenzaría con estas líneas:

```
<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
```

Ese sería el primer cambio a realizar a la hora de generar un fichero VNX.

2- Seguida de estas dos líneas en un fichero VNUML, siempre se comenzaría la descripción del escenario con: <*vnuml*> y finalizaría el escenario con <*/vnuml*>. Eso también ha cambiado pues como se observa anteriormente la primera línea comienza con <vnx … > seguidamente comenzará la descripción del escenario y terminará el fichero VNX con <*/vnx*>.

3- La etiqueta <*version*> se mantiene, pero los valores admitidos cambian, estos valores son tomados del menú archivo propiedades de la interfaz, figura 3.1.

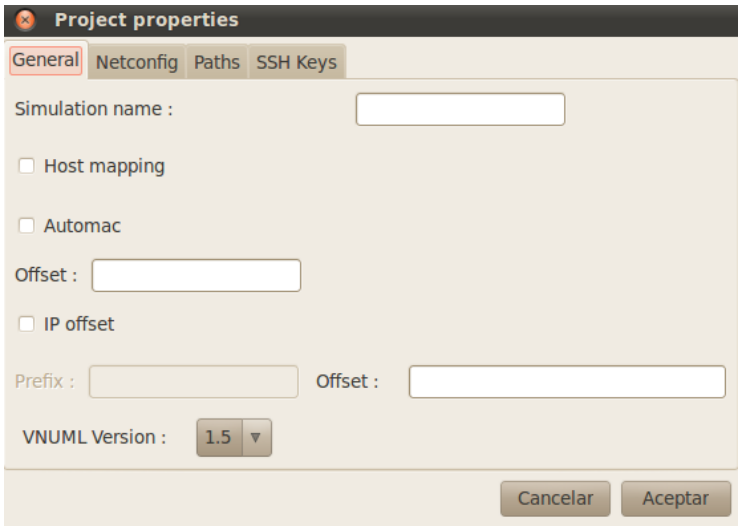


Figura 3.1. Ventana propiedades VNUMLGUI.

Según esta figura los valores permitidos son 1.5 y 1.6, para la nueva interfaz habría que cambiarlo por el valor 2, que es la versión actual de VNX.

4- En la figura 3.1 se observa además varios cambios con respecto a el nombre del escenario, en VNUML, la etiqueta que lo definía era <*simulation_name*>, para VNX cambia por <*scenario_name*>. Esto tendría que reflejarse tanto en el XML generado como en la ventana de propiedades.

5- Continuando con la figura 3.1 IP Offset ya no se utiliza en VNX con lo cual habría que eliminarlo tanto del XML como de la parte gráfica, además habría que reubicar <*host_mapping*>, puesto que esta etiqueta estaría vinculada a <*vm_mgmt*>.

6- La interfaz gráfica VNUMLGUI genera la configuración ssh con el siguiente formato: <*ssh_key version="2"*>*/home/ivan/key*</*ssh_key*>, esto ha cambiado para VNX.

7- Introducir una ventana gráfica para la etiqueta **<***vm_mgmt***>**, pues VNUMLGUI no la tiene incluida. Incluir atributos y etiquetas vinculadas; type, network, mask, offset, <*host_ mapping*> y <*mgmt_net*>.

8- Se debe modificar la forma en que se define las máquinas virtuales, ya que al introducir nuevas tecnologías como libvirt y dynamips, hay que adicionar nuevos

atributos como type, subtype y os. Además para definir la forma en que se ejecutan los comandos habría que incluir el atributo exec_mode.

9- En la etiqueta <*filesystem*> hay que eliminar el valor copy para el atributo type, ya que fue eliminada desde la version 1.7.

10- En la etiqueta <*mem*>, se sustituirán los valores para que estén dados en unidades de Megas y Gigas y no Megas y Kilo, como están en la versión de VNUMLGUI.

11- Se debe corregir la ubicación de las etiquetas ya que algunas como <*shell*>, <*basedir*>, eran ubicadas por VNUMLGUI dentro de global, ahora deben situarse dentro de la etiqueta <*vm_defaults*>, estas etiquetas son:

- <*filesystem*>
- <*kernel*>
- <*shell*>
- <*basedir*>

12- La etiqueta <*console*> ha cambiado la forma en que se define en VNX, hay que corregirlo tanto en la parte gráfica como en la generación del XML.

13- Hay que cambiar la forma en que se defina la dirección IP, antes en VNUML se hacia; <*ipv4 mask="24">10.0.0.1</ipv4*>; en VNX hay que hacerlo; <*ipv4>10.0.0.1/24</ipv4*>. Además en la declaración de las interfaces de una máquina virtual hay que incluir que siempre que sea dynamips se le debe adicionar el atributo name a la interfaz.

14- Para la declaración de las rutas VNUMLGUI genera el XML con este formato:

<*route type="inet" gw="10.0.0.1">0.0.0.0/0</route*>

En VNX cambia por lo que tendría que permitir valores como estos:

<*route type="ipv4" gw="10.0.0.1">default</route*>

15- El formato de <*filetree*> y <*exec*> ha cambiado, se debe corregir tanto en la interfaz gráfica como en la generación del XML.

16- Para las etiquetas relacionadas con el host, se deben modificar algunos atributos especificados anteriormente para las máquinas virtuales, como es el caso de la declaración de dirección IP, rutas y exec.

3.2 VNUMLGUI. Editor gráfico para VNUML

VNUMLGUI es una interfaz gráfica que permite la creación, edición, simulación de escenarios de simulación para VNUML. En este epígrafe se describirá brevemente como ha sido programada la interfaz para más adelante poder comprender los cambios realizados para la adaptación de la interfaz para VNX.

VNUMLGUI sigue un modelo orientado a objetos, en el que cada elemento de la simulación es representado por una clase. En particular se modificarán las clases Host, Router y Switch. La primera contiene todo lo referente a la configuración del host, la segunda a las maquinas virtuales y la tercera a las redes. Además se modificará el main del programa.

Las subrutinas donde se realizarán más cambios son las siguientes:

sub dump_xml(); en ella es especificado el formato de la generación del código XML. Esta función está presente en todas las clases.

sub setup_properties_dialog() y *sub apply_properties_dialog()*; estas funciones son las encargadas de la configuración de la ventana propiedades de un escenario de simulación, en la primera son declaradas todas las valores de un escenario y posteriormente son aplicados con la segunda función y guardados en variables.

sub setup_preferences_dialog() y *sub apply_preferences_dialog()*; son las encargadas de la configuración de valores iniciales para la creación de nuevos proyectos de simulación, en ellas son declaradas valores comunes como el parser de vnuml, la ruta del DTD. Y posteriormente son guardados en una carpeta local en el equipo. La primera es utilizada para declarar los valores y la segunda para aplicarlos y guardarlos en variables.

sub open_file(); esta subrutina es muy importante ya que es utilizada para cargar un XML por la herramienta gráfica, en ella el fichero XML es parseado, donde son asignados los atributos a sus respectivas variables para la generación de una topología gráfica, para ello es utilizado un algoritmo donde a cada elemento de red se le asignan variables X y Y, para su posterior construcción de su topología gráfica.

sub dialog_run(); esta función existe en cada clase (Router, Host, Switch), con ella son declarados los valores de cada elemento, según sea el caso. Y una vez se le de aceptar a la ventana, los valores son guardados en sus respectivas variables.

sub simulation_build(); es utilizada para la simulación del escenario en ella se ejecuta el comando correspondiente para la ejecución de un escenario de red, una vez aplicado el programa pasa a otro estado donde los elementos no pueden ser modificados.

sub simulation_release(); se utiliza para detener la simulación de un escenario de red, en ella es pasado el comando a VNUML para realizar el apagado de las maquinas virtuales

y demás elementos del escenario, lo que permite que el programa pase a otro estado donde vuelven a ser editable los elementos que lo conforman. Además si se le suministra el parámetro force a esta función el comando se transforma en la destrucción del escenario, esto es muy importante por si el programa VNUML por alguna razón se queda bloqueado, la única forma de volver a realizar la simulación es destruyendo el escenario.

sub simulation_cleanup(); esta función permite la limpieza del directorio de simulación.

sub simulation_execute(); con esta función es posible la ejecución de los comandos una vez iniciados todos los elementos de un escenario de red.

Otro aspecto importante a señalar es que este programa sigue el modelo de máquinas de estado, en donde se indican los diferentes estados posibles, y cual estado le sigue al otro (11). Los estados en los que se puede encontrar la simulación son:

- Void: Este es el estado inicial, todavía no se ha realizado ninguna simulación.
- Clean: La simulación ha sido salvada, pero no modificada.
- Dirty: La simulación ha sido modificada, y aún no se ha salvado.
- Building: La simulación está siendo construida por el parser, pero aún no ha concluido.
- Running: La simulación ha sido ya construida.
- Execing: La simulación está en proceso de ejecución de los comandos especificados.
- Releasing: La simulación ha sido detenida.

La definición de estados para la simulación resulta muy útil, pues mediante ella es posible que una vez ejecutada la simulación, no se permita la modificación de ningún parámetro de los elementos del escenario. Además es posible la modificación de los colores de los elementos de red, al realizar la transición entre estados. Esto constituye una ayuda visual para el usuario para el conocimiento a simple vista de en que estado se encuentra el escenario de red.

3.3 Cambios realizados para la creación de VNXGUI

En este epígrafe mediante el uso de las diferencias planteadas en el anterior se realizará una breve explicación de los cambios realizados en el código de VNUMLGUI para el desarrollo de la interfaz VNXGUI.

Primeramente se debe partir de que existen dos archivos principales entre los que conforman el programa VNUMLGUI, el archivo ***"vnumlgui"*** y ***"vnumlgui.glade"***. El primero es el programa en si creado con el lenguaje de programación Perl, el cual contiene todas las funciones que manejan a la interfaz gráfica GTK. El segundo es el diseño de la interfaz gráfica GTK creada en la herramienta de diseño de interfaces gráficas GLADE. Por lo que los cambios se centrarán en estos dos archivos.

Comenzando con la diferencia número uno, cuando se crea un escenario nuevo en VNUMLGUI, el programa genera automáticamente partes del XML como muestra el anexo 10. Y comienza específicamente por las líneas:

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd">
```

Estas líneas debemos cambiarlas por las siguientes:

```
<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="/usr/share/xml/vnx/vnx-2.00.xsd">
```

Como el cambio a realizar no implica una modificación de la interfaz gráfica, sino más bien un cambio en el texto que se genera, se revisará el programa y se buscará específicamente la función que genera el archivo XML. Esta función se encuentra en el programa con el nombre; *sub dump_xml()*, y el fragmento de código que especifica las dos primeras líneas son:

```
my $text = $CONFIG{'VNUML'}{'XMLHEAD'} . "\n";
  $text .= '<!DOCTYPE vnuml SYSTEM "' . $CONFIG{'VNUML'}{'DTD'} . '">' . "\n";
```

Donde la variable *$CONFIG{'VNUML'}{'XMLHEAD'}* es declarada en otra función *sub init_config()*, donde son inicializados ciertos parámetros de configuración del programa, la línea de código que lo muestra es:

```
$CONFIG{'VNUML'}{'XMLHEAD'} = '<?xml version="1.0" encoding="UTF-8"?>';
```

Para el diseño de la nueva interfaz VNXGUI se modifican dichas líneas quedando de la siguiente forma:

```
my $text = $CONFIG{'VNUML'}{'XMLHEAD'} . "\n";
  $text .=  '<vnx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"'. "\n";
  $text .= "\t".'xsi:noNamespaceSchemaLocation="' . $CONFIG{'VNUML'}{'DTD'} . '">'. "\n";
```

O sea se mantiene la primera línea, y modificamos las otras dos con el texto que se quiere generar. Finalmente la variable *$CONFIG{'VNUML'}{'DTD'}* especifica la ruta del fichero XSD, esta se puede declarar desde la interfaz gráfica en el menú editar, preferencias, en la pestaña general como muestra la figura 3.2.

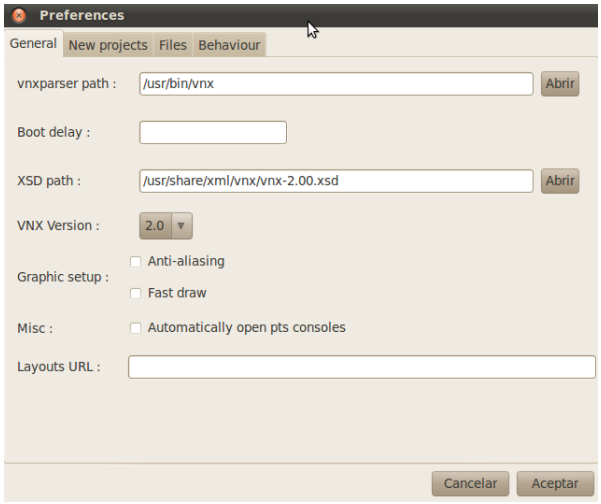


Figura 3.2. Menú preferencias de la interfaz VNXGUI.

Para la modificación de la diferencia número dos, el cambio a realizar es igual que el anterior. En la función *sub dump_xml()* la línea que comienza con <*vnuml*> es:

$text .= "<vnuml>\n\t<global>\n";

Y finaliza con *</vnuml>*:

$text .= "</vnuml>\n";

Puesto que en el primer cambio se adicionó la etiqueta *<vnx ...* solo habría que modificar la primera línea, eliminando <*vnuml*> quedando así:

$text .= "\t<global>\n";

Y la segunda habría que modificarla de la siguiente forma:

$text .= "</vnx>\n";

La diferencia número tres implica un cambio en la versión, ya que la interfaz gráfica VNUMLGUI trabaja con la versión 1.5 y 1.6 de VNUML, y la nueva interfaz gráfica trabajará con la versión 2.0 de VNX, con lo cual se debe cambiar tanto en la interfaz gráfica como en el programa.

VNUMLGUI para la declaración de la versión tiene un combobox y en el programa la variable *$CONFIG{'VERSION_INDEXES'}* esta declarada como:

```
'VERSION_INDEXES' => { '1.5' => 0,
                       '1.6' => 1, },
```

El cambio a realizar consiste en asignar a la variable *$CONFIG{'VERSION_INDEXES'}* solo el valor 2.0, para ello se modificará la línea anterior de la siguiente manera:

```
'VERSION_INDEXES' => { '2.0' => 0 },
```

Y además se realizarán cambios en los combobox del menú propiedades y preferencias, asignando solo el valor 2.0. Al ser un único valor no es tan importante que la forma de escoger la versión en la interfaz gráfica sea un combobox, pero aun así se mantiene por si en el futuro se adicionan nuevas versiones.

La modificación de la diferencia número 4 es muy parecida a la primera y segunda, puesto que se debe volver a modificar la función *sub dump_xml()* específicamente la línea vinculada con la etiqueta <*scenario_name*>. En VNUMLGUI la etiqueta utilizada era <*simulation_name*>, y la línea que la especificaba en el código es:

```
$text .= "\t\t<simulation_name>" . $STORE{'SIMULATION'}{'NAME'} . "</simulation_name>\n";
```

Donde la variable *$STORE{'SIMULATION'}{'NAME'}* toma el nombre que se le asigna al proyecto, o en el menú propiedades de la entrada de texto simulation name. Para realizar el cambio a VNXGUI se modifica la línea de código quedando de la siguiente forma:

```
$text .= "\t\t<scenario_name>" . $STORE{'SIMULATION'}{'NAME'} . "</scenario_name>\n";
```

Y además se modifica la etiqueta en el menú propiedades de simulation name a scenario name.

Para corregir la diferencia 5, puesto que es la eliminación de código solo hay que localizar lo perteneciente a IP offset:

En la función *sub apply_properties_dialog()*, donde se asignan los valores a las variables contenidas en la ventada propiedades, se eliminará el código:

```
if ($STORE{'GLADEXML'}->get_widget('ppdChkIPOffset')->get_active()) {
   $STORE{'SIMULATION'}{'IP_OFFSET'} = $STORE{'GLADEXML'}-
>get_widget('ppdEntIPOffsetOffset')->get_text();
   $STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'} = $STORE{'GLADEXML'}-
>get_widget('ppdEntIPOffsetPfix')->get_text();
  } else {
   delete($STORE{'SIMULATION'}{'IP_OFFSET'});
   delete($STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'});
  }
```

En la función *sub create_new()* donde se inicializan todos los valores de un proyecto nuevo, se eliminan los pertenecientes a IP Offset:

```
$STORE{'SIMULATION'}{'IP_OFFSET'} = $CONFIG{'VNUML'}{'IP_OFFSET'} if
($CONFIG{'VNUML'}{'IP_OFFSET'});
```

```
   $STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'} = $CONFIG{'VNUML'}{'IP_OFFSET'} if
($CONFIG{'VNUML'}{'IP_OFFSET_PREFIX'});
```

En la función *sub dump_xml()*, que se encarga de la generación del XML, se elimina:

```
  if (defined $STORE{'SIMULATION'}{'IP_OFFSET'}) {
    $text .= "\t\t<ip_offset";
    $text .= " prefix=\"" . $STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'} . "\"" if
($STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'});
    $text .= ">" . $STORE{'SIMULATION'}{'IP_OFFSET'};
    $text .= "</ip_offset>\n";
  }
```

En la función *sub open_file()*, la cual se encarga de cargar un fichero XML en la interfaz gráfica se elimina:

```
  my $offset_list = $doc->getElementsByTagName("ip_offset");
  if ($offset_list->getLength == 1) {
    # $ip_offset is modified only if the tag is not empty
    if (&text_tag($offset_list->item(0)) ne "" ) {
      $STORE{'SIMULATION'}{'IP_OFFSET'} = &text_tag($offset_list->item(0));
    }
    my $prefix = $offset_list->item(0)->getAttribute("prefix");
    unless ($prefix =~ /^$/) {
      $STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'} = $prefix;
    } #!
  }
```

En la función *sub setup_properties_dialog()*, que carga la ventana propiedades ya sea con sus valores inicializados o no, se elimina:

```
  if (defined($STORE{'SIMULATION'}{'IP_OFFSET'})) {
    $STORE{'GLADEXML'}->get_widget('ppdChkIPOffset')->set_active(TRUE);
    if (defined $STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'}) {
      $STORE{'GLADEXML'}->get_widget('ppdEntIPOffsetPfix')-
>set_text($STORE{'SIMULATION'}{'IP_OFFSET_PREFIX'});
    }
#   $STORE{'GLADEXML'}->get_widget('ppdLblIPOffsetOffset')->set_active(TRUE);
    if ($STORE{'SIMULATION'}{'IP_OFFSET'}) {
      $STORE{'GLADEXML'}->get_widget('ppdEntIPOffsetOffset')-
>set_text($STORE{'SIMULATION'}{'IP_OFFSET'});
    }
  }
```

Y finalmente se debe eliminar además los widgets vinculados con estas funciones en la interfaz gráfica, si se observa la figura 3.1 son tres, un botón de casilla y dos entradas de texto.

Continuando con los cambios, la diferencia número seis sobre la generación de la etiqueta *<ssh_key>*, radica en que VNUMLGUI genera el XML con el siguiente formato:

<ssh_key version="2">/home/ivan/key.pub</ssh_key>

Y para VNXGUI el formato debe ser el siguiente:

<ssh_version>2</ssh_version>

<ssh_key>/home/ivan/key1.pub</ssh_key>

A parte hay que tener en cuenta que para VNUML la versión de SSH por defecto era la 1, pero en VNX es la 2. Además no hay que realizar cambios en la interfaz gráfica sino en la forma de generar el XML, por lo que se modificará el código Perl. El código empleado en VNUMLGUI:

```
  $text .= "\t\t<ssh_key";
  $text .= " version=\"" . $STORE{'SIMULATION'}{'SSH_VERSION'} . "\"" if (defined
$STORE{'SIMULATION'}{'SSH_VERSION'}
                                                  and $STORE{'SIMULATION'}{'SSH_VERSION'} > 0);
  $text .= ">" . $STORE{'SIMULATION'}{'SSH_KEYS'}[0] . "</ssh_key>\n";
```

Y quedará modificado en VNXGUI así:

```
if (defined $STORE{'SIMULATION'}{'SSH_VERSION'} and $STORE{'SIMULATION'}{'SSH_VERSION'}
== 1) {
  $text .= "\t\t<ssh_version";
  $text .= ">" . $STORE{'SIMULATION'}{'SSH_VERSION'} . "</ssh_version>\n";
  }
if (defined $STORE{'SIMULATION'}{'SSH_KEYS'}) {
  $text .= "\t\t<ssh_key";
  $text .= ">" . $STORE{'SIMULATION'}{'SSH_KEYS'}[0] . "</ssh_key>\n";
  }
```

O sea lo que se hace es separar las etiquetas y no mantenerlas en una misma línea como hace VNUMLGUI.
La diferencia numero siete consiste en que VNUMLGUI no tiene incluida la etiqueta *<vm_mgmt>* para la red de gestión, con lo cual se debe introducir en la nueva interfaz VNXGUI. Para ello se comenzará con la modificación de la interfaz gráfica. VNUMLGUI poseía un botón de casilla para la etiqueta *<host_mapping>*, este botón se debe vincular a la parte gráfica de la red de gestión, junto a todos los parámetros que intervienen. Para ello se crea una nueva pestaña en la ventana de propiedades, como muestra la figura 3.3.

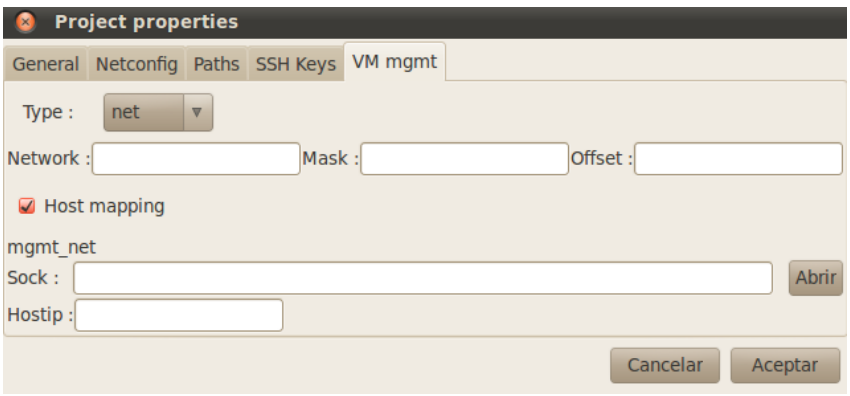


Figura 3.3. Pestaña de vm_mgmt para VNXGUI.

En ella se pueden observar los diferentes atributos y etiquetas hijas vinculadas a <*vm_mgmt*>. El combobox type específica el tipo de red; net, private, none. Y posee una función vinculada a un manejador del widget combobox, la cual según el tipo de red escogida, marca o desmarca los atributos que pueden ser escogidos para cada una.

A continuación se muestra el código empleado (por el gran tamaño del código solo se mostrarán fragmentos):

La función que se encarga de manipular el manejador *on_ppdtype_changed*, para marcar y desmarcar los demás atributos según el valor escogido en el combobox es *sub on_ppdtype_changed()*, en dicha función se plantean condiciones mediante if, según el valor activado utilizando:

```
$STORE{'GLADEXML'}->get_widget('ppdtype')->get_active()
```

Y se activan, desactivan, marcan o desmarcan mediante el uso de:

```
$STORE{'GLADEXML'}->get_widget('ppdoffset')->set_sensitive(TRUE);
        $STORE{'GLADEXML'}->get_widget('ppdChkHostmapping')->set_active(TRUE);
```

Para la generación del XML, una vez rellenados los valores en la pestaña vm_mgmt de la ventana propiedades y al darle aceptar, la función *sub apply_properties_dialog()* se encarga de introducir dichos valores en variables que posteriormente serán visualizados mediante la función *sub dump_xml()* .

Las variables utilizadas son;

```
$STORE{'SIMULATION'}{'TYPE'}
$STORE{'SIMULATION'}{'VNET'}
$STORE{'SIMULATION'}{'VMASK'}
$STORE{'SIMULATION'}{'VOFFSET'}
$STORE{'SIMULATION'}{'VSOCK'}
$STORE{'SIMULATION'}{'VHOSTIP'}
```

Según el valor que tome type algunas serán usadas y otras no. En todas las funciones hay algoritmos condicionales que especifican las acciones a realizar según el tipo de red. La otra función utilizada es *sub open_file()*, en la cual según los valores encontrados a la hora de abrir un XML, le asigna los valores a las variables que posteriormente serán generadas en la interfaz gráfica.

La diferencia número ocho consiste en los nuevos atributos que definen a una máquina virtual, estos atributos son; type, subtype, os y exec_mode, para ello se debe modificar tanto la interfaz gráfica como el código del programa. La interfaz gráfica quedaría como muestra la figura 3.4:

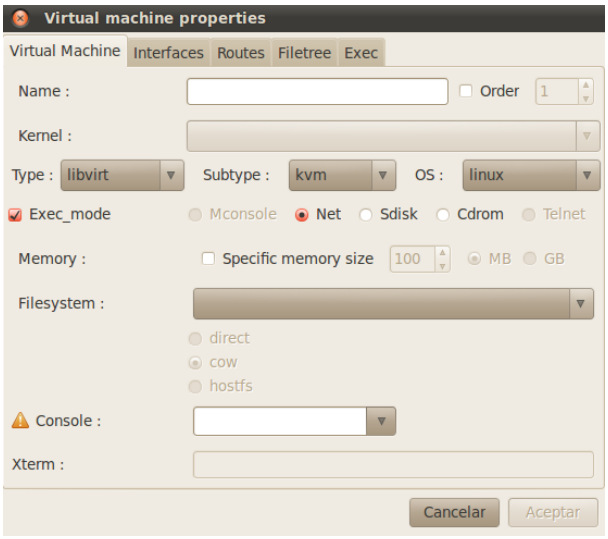


Figura 3.4. Ventana de una máquina virtual enVNXGUI.

Esta modificación consiste en que según el tipo de máquina que sea, es escogido el subtipo y el sistema operativo, a su vez si se activa el botón de casilla Exec_mode, son marcados y desmarcados los valores de exec_mode, según el tipo de máquina virtual.

Para ello se utiliza el mismo método anterior, el combobox type tiene un manejador que responde a la función sub on_vmtype_changed(). Esta subrutina es ejecutada cada vez que se realiza un cambio en el combobox type. La subrutina contiene funciones como:

```
if ($STORE{'GLADEXML'}->get_widget('vmtype')->get_active() == 0) {
$STORE{'GLADEXML'}->get_widget('vmsubtype')->get_model()->clear();
$STORE{'GLADEXML'}->get_widget('vmos')->get_model()->clear();
my (@strings2,@strings3) = ();
foreach (@strings2) {
$STORE{'GLADEXML'}->get_widget('vmsubtype')->append_text ($_);
}
foreach (@strings3) {
$STORE{'GLADEXML'}->get_widget('vmos')->append_text ($_);
}
$STORE{'GLADEXML'}->get_widget('vmsubtype')->set_sensitive(FALSE);
```

Las funciones anteriores es un fragmento del código utilizado, en ella es posible observar como se realiza el chequeo de que valor esta activo en el combo y según eso limpiar y asignar nuevos valores al resto de los combobox, en este ejemplo en particular los combobox subtype y os quedan vacíos, ya que es para la opción en que type toma el valor de uml.

Además se adicionan las subrutinas *sub on_vmos_changed()* y *sub on_vmexec_mode_toggled()*, la primera se activa cuando es modificado el combobox os, y define cuales valores de exec_mode se activan según el tipo de sistema operativo. La segunda se ejecuta cuando es activado el botón de casilla Exec mode, y siempre chequea antes, que valores tienen los combobox type, subtype, y os, según los valores que tengan, activa o desactiva los valores de exec_mode.

Se realizan modificaciones en la función *sub dialog_run()* en ella una vez escogidos los valores como muestra la figura 3.4, y se le da clic al botón aceptar, se asignan valores a las variables:

```
$self->{'type'};
$self->{'subtype'};
$self->{'os'};
$self->{'exec_type'}
```

Y una vez inicializadas las variables, mediante el siguiente código es generado el formato XML:

```
$xml .= " type=\"" . $self->{'type'} . "\"";
  $xml .= " subtype=\"" . $self->{'subtype'} . "\"" if (defined $self->{'subtype'});
  $xml .= " os=\"" . $self->{'os'} . "\"" if (defined $self->{'os'});
  $xml .= " order=\"" . $self->{'order'} . "\"" if $self->{'order'};
  $xml .= " exec_mode=\"" . $self->{'exec_type'} . "\"" if $self->{'exec_type'};
  $xml .= ">\n";
```

Finalmente se adicionan nuevas funciones a la subrutina *sub open_file()*, ellas permiten asignar valores a las variables vinculadas con estas etiquetas y atributos, al abrir un archivo VNX mediante la interfaz gráfica VNXGUI.

El otro cambio a realizar es en la misma ventana anterior, en VNUML, se podían declarar 4 tipos de filesystem, para VNX, esto ha cambiado, el valor copy se ha eliminado, con lo cual se debe eliminar tanto de la interfaz gráfica como del programa.

Para ello se elimina el botón de radio copy mostrado en la figura 3.5:

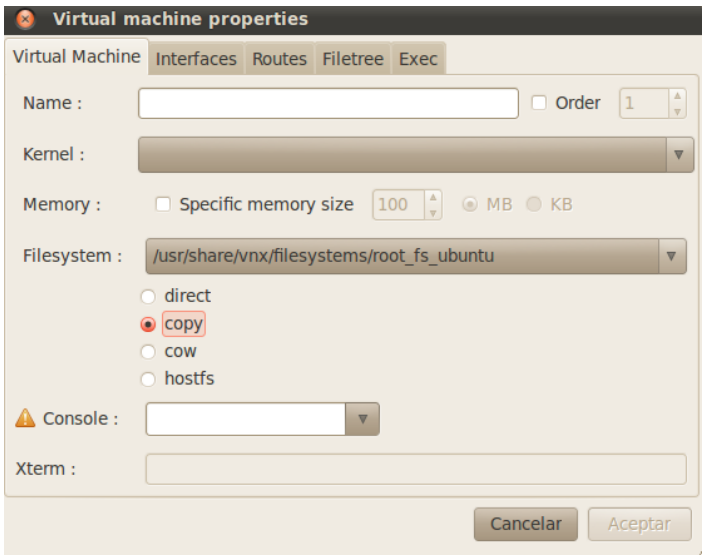


Figura 3.5. Ventana gráfica de una máquina virtual en VNUMLGUI.

Además se elimina el código correspondiente al botón, este se encuentra en la subrutina donde se inicializan y se asignan los valores a las variables de las maquinas virtuales, esta función es *sub dialog_run()* y el código que hay que corregir es:

```
if (defined($self->{'filesystem_type'}) and $self->{'filesystem_type'}) {
  foreach my $type (qw/direct copy cow hostfs/) {
    $STORE{'GLADEXML'}->get_widget('rdRdoFs'.ucfirst($type))->set_sensitive(TRUE);
    if ($self->{'filesystem_type'} eq $type) {
          $STORE{'GLADEXML'}->get_widget('rdRdoFs'.ucfirst($type))->set_active(TRUE) }
          }
}
```

O sea que según la variable *$type*, marca y activa el widget correspondiente, por lo que si se elimina el botón radio copy, se debe eliminar a copy de la variable type, quedando la línea modificada de la siguiente forma:

```
foreach my $type (qw/direct cow hostfs/) {
```

Esto se realiza en todos las subrutinas donde se encuentre el código correspondiente al botón radio copy.

Si se observa la figura 3.5, las unidades en que se expresa la memoria de una máquina virtual esta dada en kilos y megas, por lo que se realizará el cambio de Kilos por Gigas, en la parte gráfica es muy fácil solo sería cambiar la etiqueta KB por GB, y en el programa, habría que cambiar la forma en que se declara la memoria, o sea que VNXGUI debe generar en el XML, la memoria de la siguiente manera: *<mem>256M</mem>* (M para megas o G para gigas).

Para ello se modifican los valores que pueden tomar las unidades de memoria, específicamente la variable:

```
$self->{'umemory'} = 'M'
$self->{'umemory'} = 'G'
```

Para la corrección de la diferencia 11, se debe cambiar la forma en que se genera el fichero XML, con lo cual se adiciona una condición de que siempre que se defina algún parámetro de la pestaña Default VM, ver figura 3.6, estos van a ser ubicados dentro de la etiqueta *<vm_defaults>* del archivo VNX. Para ello se introducen las siguientes líneas en la función *sub dump_xml()*:

```
$text .= "\t\t<vm_defaults>\n";
$text .= "\t\t</vm_defaults>\n";
```

Entre dichas líneas de código, estará definido el código que genera los valores de los atributos definidos en la figura 3.6.

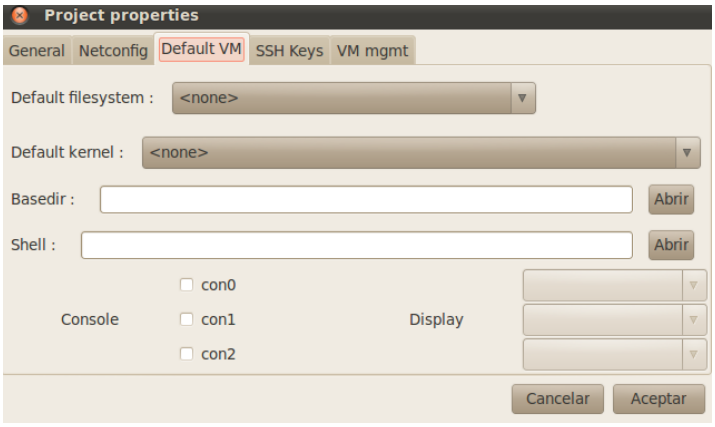


Figura 3.6. Parámetros por defecto en ventana propiedades de VNXGUI.

Como se observa en la figura se adiciona la etiqueta <*console*> en <vm_defaults>. Y como todas estas etiquetas son hijas de <*vm_defaults*>, se modifica la subrutina *sub open_file()* con la siguiente línea:

```
my $vm_defaults = $doc->getElementsByTagName('vm_defaults');
```

La diferencia numero 13 es también una simple edición de la función *sub dump_xml()*, VNUMLGUI, al generar la dirección IP de una máquina virtual lo hace con este código:

```
my ($i4,$m4) = split('/',$ip);
$xml .= " mask=\"" . $m4 . "\"" if $m4;
```

Primeramente se separa la dirección IP de la máscara guardando la IP en *$i4* y la máscara en *$m4*. Y luego la máscara es definida con el formato de la segunda línea, quedando de la siguiente forma: <*ipv4 mask="24">10.0.0.1</ipv4*>. VNXGUI debe generar la dirección IP de otra manera: <*ipv4>10.0.0.1/24</ipv4*>. Para ello se elimina la línea que permite la separación de la dirección IP y la máscara y se define la dirección IP sin separar con el siguiente código:

```
$xml .= "\t\t\t<ipv4>" . $ip . "</ipv4>\n";
```

Lo mismo se realiza para la función *sub dump_xml()* del host.

Además se introduce el campo name, como muestra la figura 3.7 en la pestaña para la configuración de las interfaces de una máquina virtual, este campo es usado solo para cuando la máquina es tipo dynamips.

Para que name sea generado en el archivo XML, se adiciona líneas de código a la subrutina *sub dump_xml()*, asi como a *sub open_file()*, para asignar el valor correspondiente a name una vez que es leído el archivo XML por VNXGUI.

Quedando una interfaz de una máquina tipo dynamips en VNX de la siguiente forma:

<if id="1" net="Net0" name="e0/1">
<if id="2" net="Net1" name="e0/2">

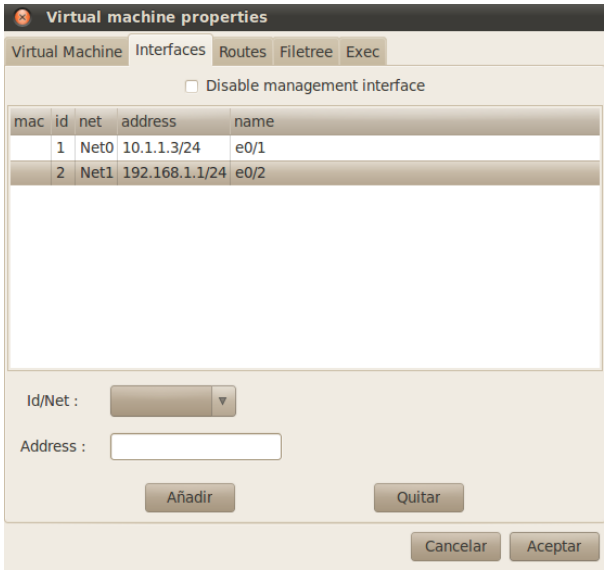


Figura 3.7 Interfaces de red de una máquina virtual en VNXGUI.

Se ha modificado además todo lo referente a la etiqueta *<console>*, en VNUMLGUI dicha etiqueta era generada dentro de *<boot>*, esto se hacía desde la interfaz gráfica de la figura 3.5. Para VNXGUI, la interfaz gráfica quedaría como muestra la figura 3.8.

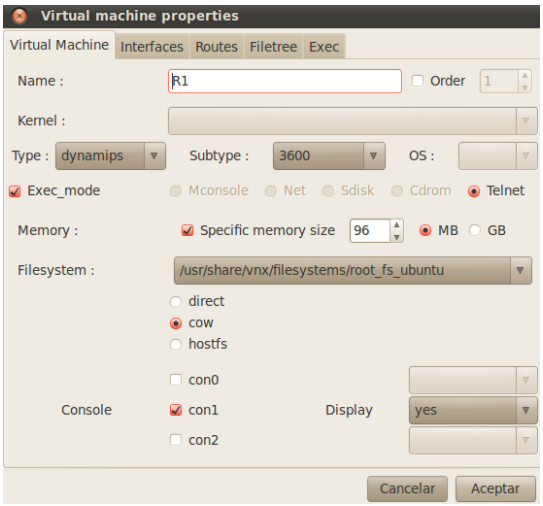


Figura 3.8. Modificación de console para VNXGUI.

Como se observa se tienen 3 botones de casilla los cuales al marcarse activan a los combobox, donde se puede asignar el valor "yes" o "no", según sea el caso, al atributo display. Y al desmarcar las casillas los combobox se desactivan. Estas funciones de marcado y desmarcado de los botones de casilla se logran a través de las subrutinas:

sub on_vmconsole0_toggled(),
sub on_vmconsole1_toggled(),

sub on_vmconsole2_toggled()

Y en cada una es definido el siguiente código:

$STORE{'GLADEXML'}->get_widget('vmcondisplay0')->set_sensitive($STORE{'GLADEXML'}->get_widget('vmconsole0')->get_active());

El código XML es generado a través de la función *sub dump_xml()*, y esta función toma las variables desde la subrutina *sub dialog_run()*, la cual una vez aceptada la ventana gráfica obtiene los valores y se los asigna a dos arreglos:

@{$self->{'consoleid'}}
@{$self->{'consoledis'}}

Además se le adiciona el código respectivo a la función *sub open_file()*, para realizar la carga del XML, por VNXGUI. Esta función también le asigna los valores a los arreglos anteriores, para que la función *sub dump_xml()* genere el código XML.

El ejemplo mostrado en la figura 3.8 donde es marcada la con1 y asignado el valor "yes" a display generaría el código XML siguiente:

<console id= "1" display= "yes" />

En cuanto a la generación de la etiqueta *<route>*, VNUMLGUI lo hacía con el siguiente formato:

<route type="inet" gw="10.1.1.3">192.168.1.0/24</route>

Y para que VNX procese el archivo XML dicha etiqueta tiene que estar en este formato:

<route type="ipv4" gw="10.1.1.3">192.168.1.0/24</route>

O sea que el cambio radica en el atributo type que para una dirección Ipv4 tiene que tener el valor "ipv4" y para una IPv6 tiene que tener este valor "ipv6".

Para ello se modifica la forma de generar el XML en la función *sub dump_xml()*. Tanto para el host como para las máquinas virtuales, se sustituyen estas líneas:

$xml .= "inet6\" ";
$xml .= "inet\" ";

Por estas:

$xml .= "ipv6\" ";
$xml .= "ipv4\" ";

Para cambiar lo relativo a *<filetree>* hay que modificar tanto la interfaz gráfica como el código del programa. VNUMLGUI genera la etiqueta *<filetree>* de la siguiente manera:

<filetree root="/directorio dest" when="start">/directorio fuente</filetree>

Esto ha cambiado en VNX, la etiqueta debe ser generada en este formato:

<filetree seq="secuencia" root="/directorio dest>/directorio fuente</filetree>

Para ello se debe modificar la pestaña de filetree de la figura 3.9.

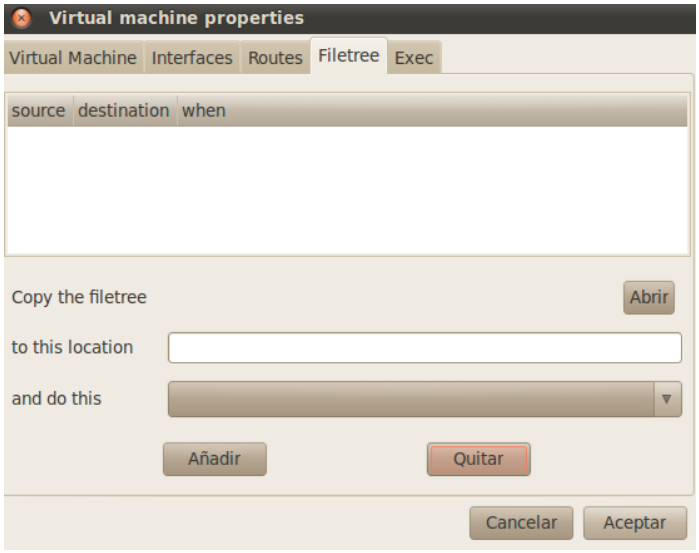


Figura 3.9. Pestaña filetree en VNUMLGUI.

El combobox que responde al atributo when se sustituye por una entrada de texto donde pueda especificarse el atributo seq. Además se sustituirá when por sequence en el campo de la tabla donde se insertan los filetree en el modo gráfico, como muestra la figura 3.10:

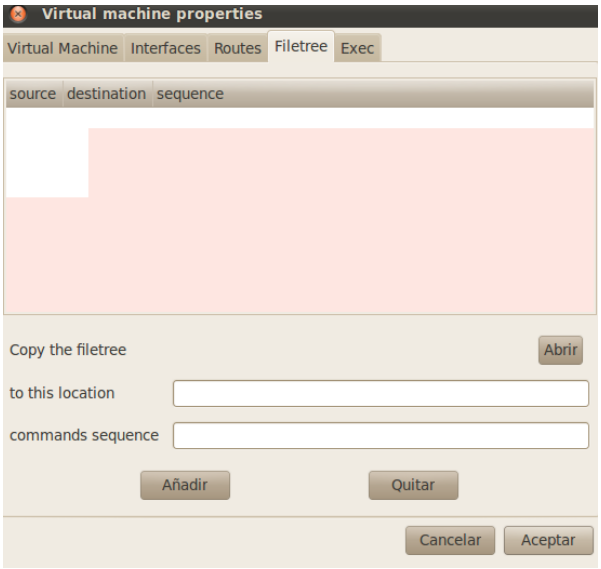


Figura 3.10. Pestaña filetree en VNXGUI.

Se modifica además el código para que genere el atributo seq, para ello se cambia la función *sub dump_xml()* y se inserta la siguiente línea de código:

$xml .= "\t\t<filetree seq=\"" . $ftree->{'sequence'};

La variable *$ftree->{'sequence'}* obtiene el valor de la función *sub dialog_run()*, una vez adicionada la etiqueta *<filetree>*, esta función obtiene los valores que están en la tabla donde se guardan las etiquetas *<filetree>*. Se corrige además la función *sub open_file()*, para que cuando detecte el atributo seq le asigne el valor a la variable *$ftree->{'sequence'}*.

La siguiente modificación es con respecto a la etiqueta *<exec>*, VNUMLGUI lo genera de la siguiente forma:

<exec seq="secuencia" type="verbatim">comando</exec>

Esta etiqueta cambia para VNX, ahora debe generarse con el siguiente formato:

<exec seq="secuencia" type="verbatim" ostype="system">comando</exec>

Como se observa se mantienen los dos primeros atributos, pero es adicionado un tercero ostype, y como este atributo puede tomar solo ciertos valores, es creado un combobox, para que se escoja el valor a adicionar, la pestaña de esta etiqueta quedaría como muestra la figura 3.11:

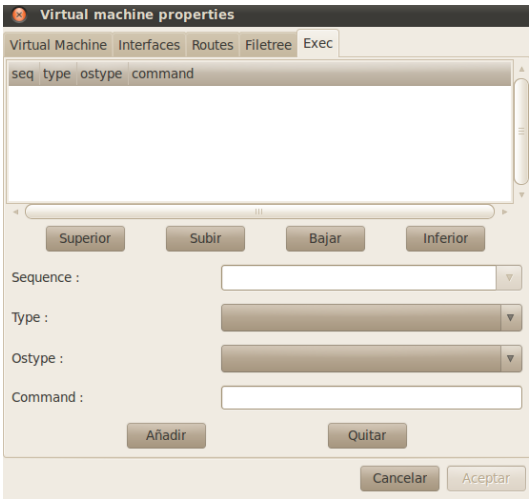


Figura 3.11. Pestaña exec para VNXGUI.

Los cambios son muy parecidos a filetree, solo hay que adicionar una nueva variable *$ex->{'ostype'}*, en la subrutina *sub dialog_run()* y en *sub open_file()* para abrir el archivo XML y finalmente se adiciona en *sub dump_xml()* la siguiente línea para la generación del XML:

$xml .= "\" ostype=\"" . $ex->{'ostype'};

Este cambio se realiza tanto para las maquinas virtuales como el host.

Son corregidas además otras funciones para la creación, apagado y destrucción del escenario de simulación, estas son:

sub simulation_build()

sub simulation_release()

La primera se encarga de la simulación del escenario, para ello se corrige el comando a ejecutar, donde VNXGUI le pasa los parámetros, como el XML a simular, y en este caso -t para la creación del escenario.

En caso de que la simulación presente algún problema se cuenta con el menú: ver, Logs, donde se puede ver los mensajes que nos pasa VNX al ejecutar los comandos pasados por VNXGUI.

La segunda subrutina apaga el escenario, suministrándole el parámetro "-d" a VNX. Y en caso de que se ejecute el menú: simulation, forced release, se le pasa el parámetro forced a la función *sub simulation_release()*, provocando que se cambie el comando a pasarle a VNX, en este caso "-P", lo cual destruye el escenario creado.

Otro cambio importante para el funcionamiento de VNXGUI es la función *sub parse_file()*, en ella es validado el XML, cuando es abierto por VNXGUI. Esto comprueba que el XML tenga una sintaxis correcta.

3.3.1 Dibujo de la topología de red

Durante la carga de un fichero XML por VNUMLGUI, como se ha comentado en epígrafes anteriores, son asignados los valores (x, y), para realizar el dibujo de la topología de red. Se observó que la herramienta gráfica, realizaba el dibujo, en el que no podían ser apreciados todos los elementos que componen el escenario de una manera adecuada, ya que estaban casi todos uno encima de otro.

Con lo que se propuso la realización de una subrutina, que mientras no fuera salvado las coordenadas del dibujo con anterioridad, se le asignaran por defecto valores (x, y), más razonables.

Esto supuso la creación de una subrutina llamada *sub get_default_layout()*, en ella, al suministrarle el parámetro que especifica la ruta del XML a cargar por VNXGUI, son ejecutados dos comandos:

```
$command1 = 'vnx2dot '. $file. '> /tmp/'.$filename.'.dot';

$command2 = 'neato -Tplain /tmp/'.$filename.'.dot 2>&1 |';
```

El primero se encarga de la conversión del fichero XML en ".dot", lo cual es importante ya que el segundo comando "neato", no acepta el XML, sino el ".dot". En este segundo comando a través de la función neato se asignan coordenadas más lógicas para la realización del dibujo de un escenario de red.

Finalmente esta función *sub get_default_layout()* devuelve un arreglo asociativo donde a cada elemento del escenario le asigna las coordenadas (x, y).

Luego es adicionada la función *sub restore_default_layout()*, en ella son asignados las coordenadas para cada clase, y finalmente *sub restore_default_layout()*, se ubica en la función *sub open_file()*, para la asignación de estas coordenadas por defecto a todos los archivos XML abiertos por VNXGUI.

3.4 Simulación de un escenario de red con VNXGUI

Para comenzar con la creación de un escenario de red en VNXGUI, se va a explicar como son creados y configurados cada elemento del escenario. Primeramente para la creación de una máquina virtual, después de crear un nuevo proyecto, se da clic derecho en la pantalla del programa, en donde se selecciona el elemento de red a adicionar, para una máquina virtual, se ejecuta la siguiente ventana:

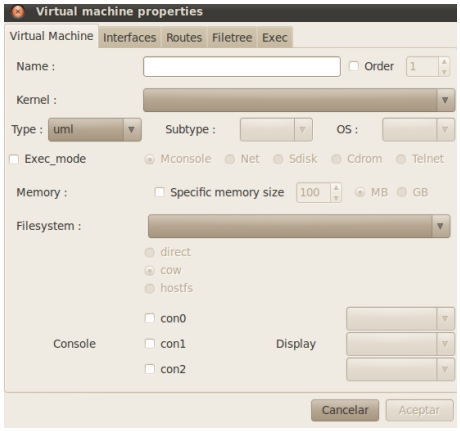


Figura 3.12. Ventana de configuración de una máquina virtual.

Como se observa en ella es posible configurarle el nombre, tipo de máquina, subtipo y sistema operativo, también es posible especificar el modo de ejecución de comandos con exec_mode, la memoria y filesystem a utilizar, así como las consolas a ejecutar.

En la figura siguiente es asignada la dirección IP de la máquina y la red a la que se conectará, para ello es necesario primero crear la red y darle clic derecho a la máquina virtual y conectar a la red deseada, una vez que se encuentren unidas, el campo Net, es actualizado con las redes que están conectadas a la máquina virtual.

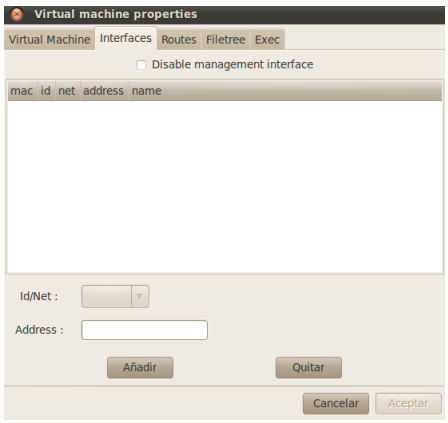


Figura 3.13. Ventana de interfaces de red de una máquina virtual.

La dirección IP debe ser adicionada en el formato; dirección ip/máscara.

En las siguientes pestañas es posible además la configuración de las rutas, filetree y exec, de cada máquina virtual.

Para la configuración de una red, basta con dar clic derecho en la pantalla al igual que una máquina virtual, la cual ejecuta la siguiente ventana:

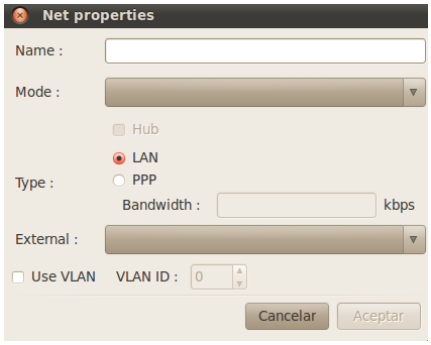


Figura 3.14. Ventana de configuración de una Red.

En ella puede ser configurada el nombre, el modo: virtual_bridge o uml_switch, asi como los demás parámetros de una red, incluso si quiere ser conectada a una interfaz de red del host con la cual pueda salir a una red externa.

Y el otro elemento es el host, mediante el mismo procedimiento puede ser adicionado al escenario, puede ser configurado, mediante la siguiente ventana gráfica:

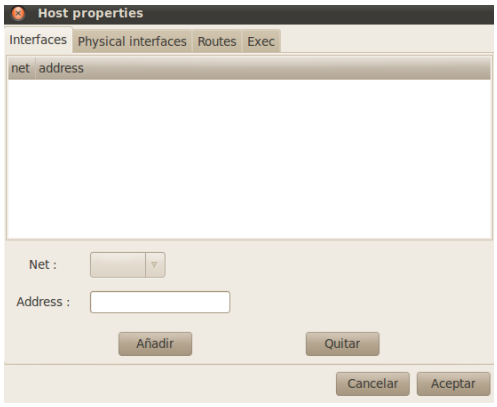


Figura 3.15. Ventana de configuración del host.

En esta ventana es posible configurar la dirección IP, al igual que una máquina virtual debe ser unida a una red para que sea activado el combobox Net. Es posible además configurar physicalif, las rutas y exec.

Para la creación de un escenario, es importante que si es la primera vez que se ejecuta VNXGUI, deben configurarse ciertos valores comunes. Como son la ruta del parser de VNX, normalmente en "/usr/bin/vnx" y la ruta del XSD, que se encuentra en "/usr/share/xml/vnx/vnx-2.00.xsd". Así como las rutas donde se encuentran los filesystem, kernel y ssh, para su posterior configuración en cada máquina virtual. Esto es posible accediendo a la ventana preferencias.

También se deben configurar las etiquetas contenidas en global, se puede realizar con la ventana propiedades:

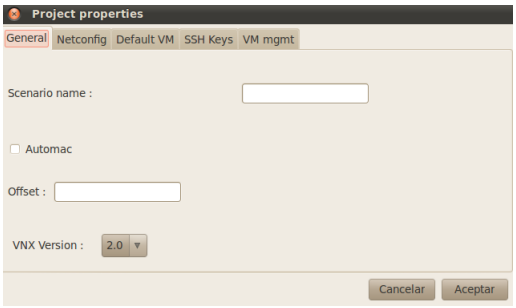


Figura 3.16. Ventana propiedades de un escenario de red.

Como se observa puede ser configurado el nombre del escenario, automac, la versión de VNX, Netconfig, las etiquetas dentro de vm_defaults, las llaves publicas SSH, y la red de gestión VM mgmt.

Ya se ha visto que es posible crear un escenario de red, configurando cada elemento e interconectándolos entre ellos, según se haya diseñado, en este trabajo se realizará la carga del escenario tutorial_ubuntu.xml, que se encuentra en /usr/share/vnx/examples/, este es uno de los escenarios que trae VNX como ejemplo. Una vez abierto el archivo genera la siguiente topología gráfica con el algoritmo de neato:

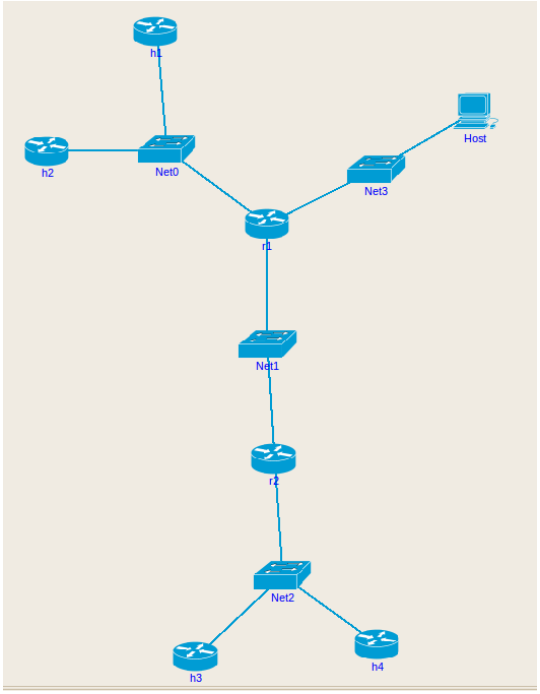


Figura 3.17. Topología gráfica por defecto de escenario de red; tutorial_ubuntu.xml.

En caso de que no sea de agrado el diseño gráfico obtenido también es posible mover manualmente cada elemento de la topología, asignándole una nueva posición a cada uno, un ejemplo de un nuevo dibujo se muestra en la siguiente figura:

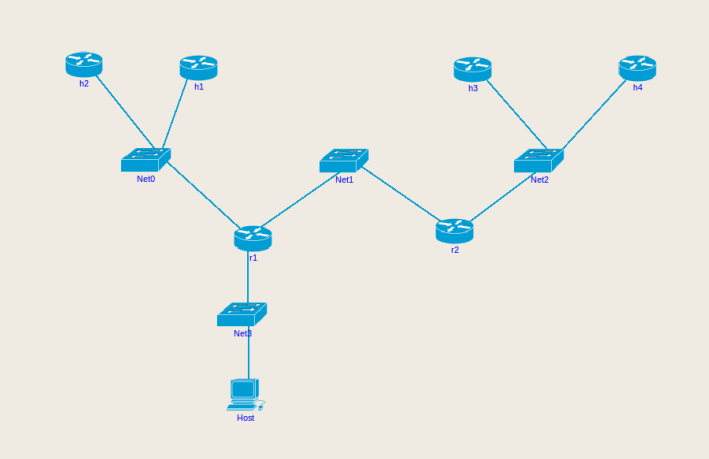


Figura 3.18. Topología gráfica personalizada de escenario de red; tutorial_ubuntu.xml.

Siempre que sea modificado los valores de las coordenadas de cada elemento, es posible salvarlas, estas son guardadas en un archivo layout, dentro de la carpeta /home/usuario.

Una vez diseñada la interfaz y modificada a nuestro gusto es posible comenzar la simulación, se puede hacer desde el menú "Simulation", o desde la barra de herramientas. Una vez iniciada, los elementos del escenario con dibujados de color verde y sin posibilidad de editarlos. Si se le da clic derecho a cada máquina virtual, se muestra un campo llamado execute, desde el cual pueden ser ejecutados los comandos declarados en cada máquina en la etiqueta <*exec*>.

Finalmente se puede liberar el escenario desde el menú "simulation realease", desde el cual se apagan las maquinas virtuales, o desde "simulation force reléase", que destruye el escenario.

Conclusiones

Este trabajo ha resultado de vital importancia para el aprendizaje de los conocimientos necesarios sobre la herramienta de virtualización de escenarios VNX. Se puede concluir que esta herramienta posibilita un ahorro de costes, tanto de infraestructura como el esfuerzo a invertir en su gestión y configuración, ya que pueden ser simulados en una máquina Linux numerosas maquinas virtuales, e interconectarlas entre si.

El estudio de las librerías GTK junto al lenguaje de programación Perl posibilitó una mayor comprensión del código de programación de la interfaz gráfica VNUMLGUI. Mediante esta interfaz fue posible la adaptación del código con el que estaba programado para la realización de la interfaz VNXGUI.

Una vez terminada la interfaz VNXGUI, se realizó la simulación del escenario "tutorial_ubuntu.xml", contenido en la carpeta example de VNX. Con un resultado positivo en la simulación. Comprobando la utilidad de una interfaz gráfica para la gestión de un escenario VNX. Al cargar un escenario VNX, mediante VNXGUI, esta muestra la topología del escenario de red, mostrando todos sus elementos e interconexiones, esto nos permite una mejor idea del escenario. Es posible además la gestión de cada elemento, donde se le pueden configurar todos sus parámetros.

Después de simular un escenario, la aplicación cambia a otro estado, donde las máquinas virtuales no pueden ser modificadas, y pueden conectarse a ellas con solo darle clic derecho a cada una de ellas. Puede ser a través de una conexión SSH, siempre y cuando se active la red de gestión (vm_mgmt) o a través de las consolas en dependencia del tipo de máquina.

Se puede decir que los objetivos fundamentales de este trabajo han sido cumplidos, no obstante la interfaz gráfica puede mejorarse aún más. Uno de los trabajos a continuar, es realizar una actualización de algunas etiquetas comunes para todas las máquinas virtuales de un escenario. Además es recomendable continuar la actualización de VNXGUI, ya que le herramienta de virtualización VNX al estar en constante evolución puede provocar que VNXGUI vuelva a quedar desfasada.

Referencias

1. VNUML-WIKI. [En línea] http://dit.upm.es/vnuml.

2. VNX. [En línea] http://dit.upm.es/vnx.

3. **Dike, Jeff.** *User Mode Linux.* Crawfordsville, Indiana : RR Donnelley, 2006. 0-13-186505-6.

4. libvirt: The virtualization API. [En línea] http://libvirt.org/.

5. Anatomy of the libvirt virtualization library. [En línea] http://www.ibm.com/developerworks/linux/library/l-libvirt/.

6. Dynamips / Dynagen Tutorial. [En línea] http://dynagen.org/tutorial.htm.

7. **Castellano, F. Javier García.** Tutorial de XML. [En línea] Marzo de 2003. http://flanagan.ugr.es/xml/.

8. Gtk2-Perl. [En línea] http://gtk2-perl.sourceforge.net/doc/pod/idx.html.

9. **Wilhelm, Stephen.** *Gtk-Perl Tutorial.* July 05, 2001.

10. Glade - A User Interface Designer. [En línea] http://glade.gnome.org/.

11. **Sofía Muñóz Vélez, Silvia Corredor Sanz y Alicia Melado Corral.** Complu6IX: Transición del Campus de la Complutense a la tecnología IPv6 - Soporte y Simulación. s.l. : UNIVERSIDAD COMPLUTENSE DE MADRID.

12. Perl Programming Documentation. [En línea] http://perldoc.perl.org/.

13. The CPAN Search Site. [En línea] http://search.cpan.org/.

14. Tutorial de Perl. [En línea] http://www.cicei.com/ocon/gsi/tutorial_perl/.

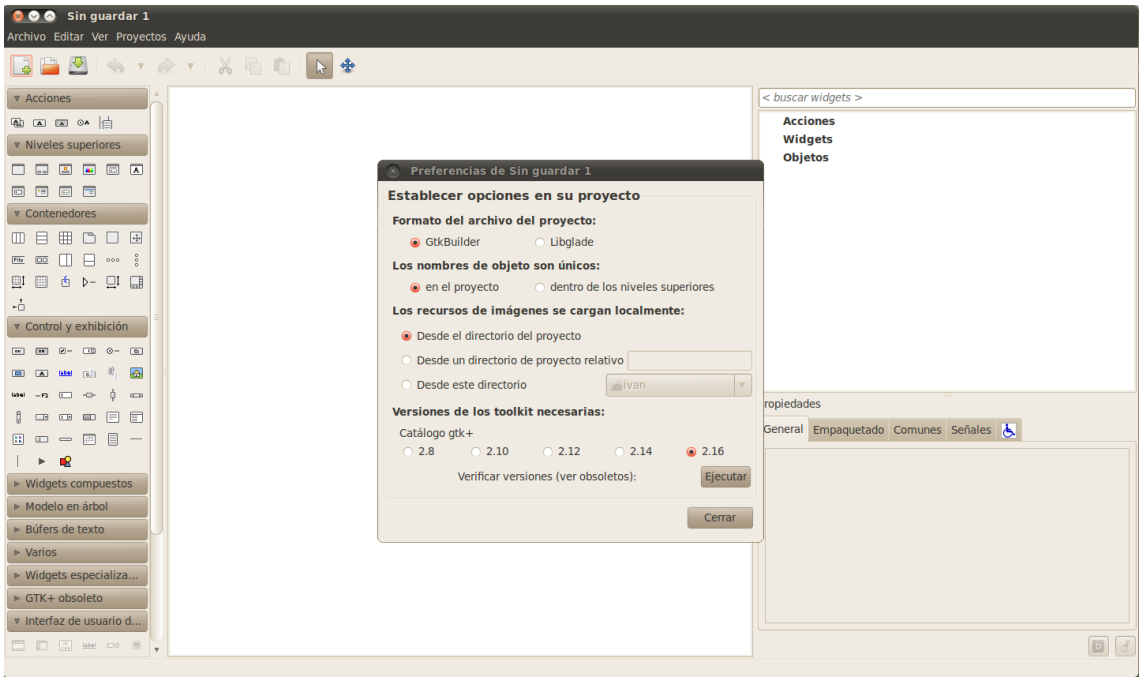
Anexos

Anexo 1. Árbol de jerarquía de clases utilizada para la implementación de los widgets:

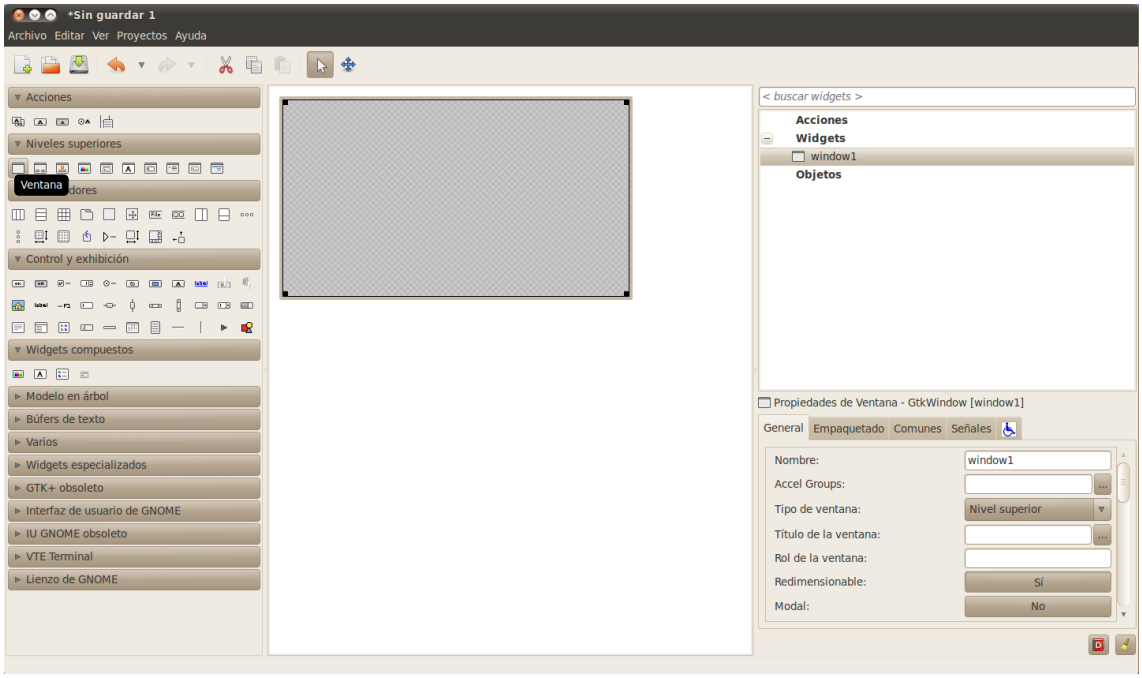
```
Gtk::Object
+-Gtk::Widget
| +-Gtk::Misc
| | +-Gtk::Label
| | | +-Gtk::AccelLabel
| | | +-Gtk::TipsQuery
| | +-Gtk::Arrow
| | +-Gtk::Image
| | +-Gtk::Pixmap
| +-Gtk::Container
| | +-Gtk::Bin
| | | +-Gtk::Alignment
| | | +-Gtk::Frame
| | | | +-Gtk::AspectFrame
| | | +-Gtk::Button
| | | | +-Gtk::ToggleButton
| | | | | +-Gtk::CheckButton
| | | | | +-Gtk::RadioButton
| | | | +-Gtk::OptionMenu
| | | +-Gtk::Item
| | | | +-Gtk::MenuItem
| | | | | +-Gtk::CheckMenuItem
| | | | | | +-Gtk::RadioMenuItem
| | | | | +-Gtk::TearoffMenuItem
| | | | +-Gtk::ListItem
| | | | +-Gtk::TreeItem
| | | +-Gtk::Window
| | | | +-Gtk::ColorSelectionDialog
| | | | +-Gtk::Dialog
| | | | | +-Gtk::InputDialog
| | | | +-Gtk::DrawWindow
| | | | +-Gtk::FileSelection
| | | | +-Gtk::FontSelectionDialog
| | | | +-Gtk::Plug
| | | +-Gtk::EventBox
| | | +-Gtk::HandleBox
| | | +-Gtk::ScrolledWindow
| | | +-Gtk::Viewport
| | +-Gtk::Box
| | | +-Gtk::ButtonBox
| | | | +-Gtk::HButtonBox
| | | | +-Gtk::VButtonBox
| | | +-Gtk::VBox
| | | | +-Gtk::ColorSelection
| | | | +-Gtk::GammaCurve
| | | +-Gtk::HBox
| | | +-Gtk::Combo
| | | +-Gtk::Statusbar
| | +-Gtk::CList
| | | +-Gtk::CTree
| | +-Gtk::Fixed
```

```
| | +-Gtk::Notebook
| | | +-Gtk::FontSelection
| | +-Gtk::Paned
| | | +-Gtk::HPaned
| | | +-Gtk::VPaned
| | +-Gtk::Layout
| | +-Gtk::List
| | +-Gtk::MenuShell
| | | +-Gtk::MenuBar
| | | +-Gtk::Menu
| | +-Gtk::Packer
| | +-Gtk::Socket
| | +-Gtk::Table
| | +-Gtk::Toolbar
| | +-Gtk::Tree
| +-Gtk::Calendar
| +-Gtk::DrawingArea
| | +-Gtk::Curve
| +-Gtk::Editable
| | +-Gtk::Entry
| | | +-Gtk::SpinButton
| | +-Gtk::Text
| +-Gtk::Ruler
| | +-Gtk::HRuler
| | +-Gtk::VRuler
| +-Gtk::Range
| | +-Gtk::Scale
| | | +-Gtk::HScale
| | | +-Gtk::VScale
| | +-Gtk::Scrollbar
| | +-Gtk::HScrollbar
| | +-Gtk::VScrollbar
| +-Gtk::Separator
| | +-Gtk::HSeparator
| | +-Gtk::VSeparator
| +-Gtk::Preview
| +-Gtk::Progress
| +-Gtk::ProgressBar
+-Gtk::Data
| +-Gtk::Adjustment
| +-Gtk::Tooltips
+-Gtk::ItemFactory
```

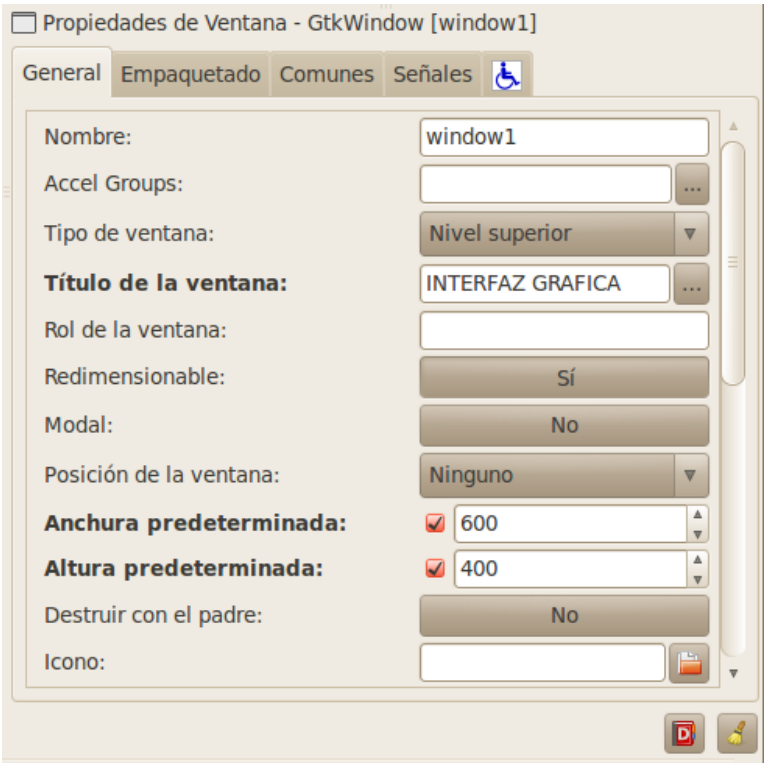
Anexo 2. Inicio de nuevo proyecto Glade:



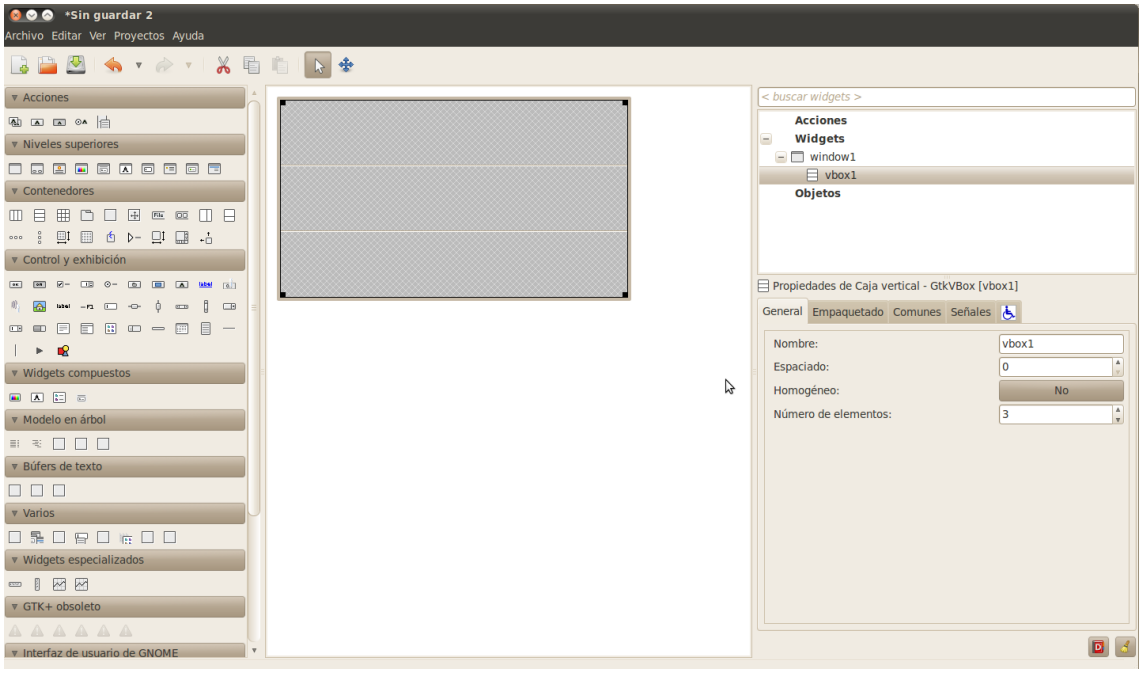
Anexo 3. Creación de una ventana como nivel superior:



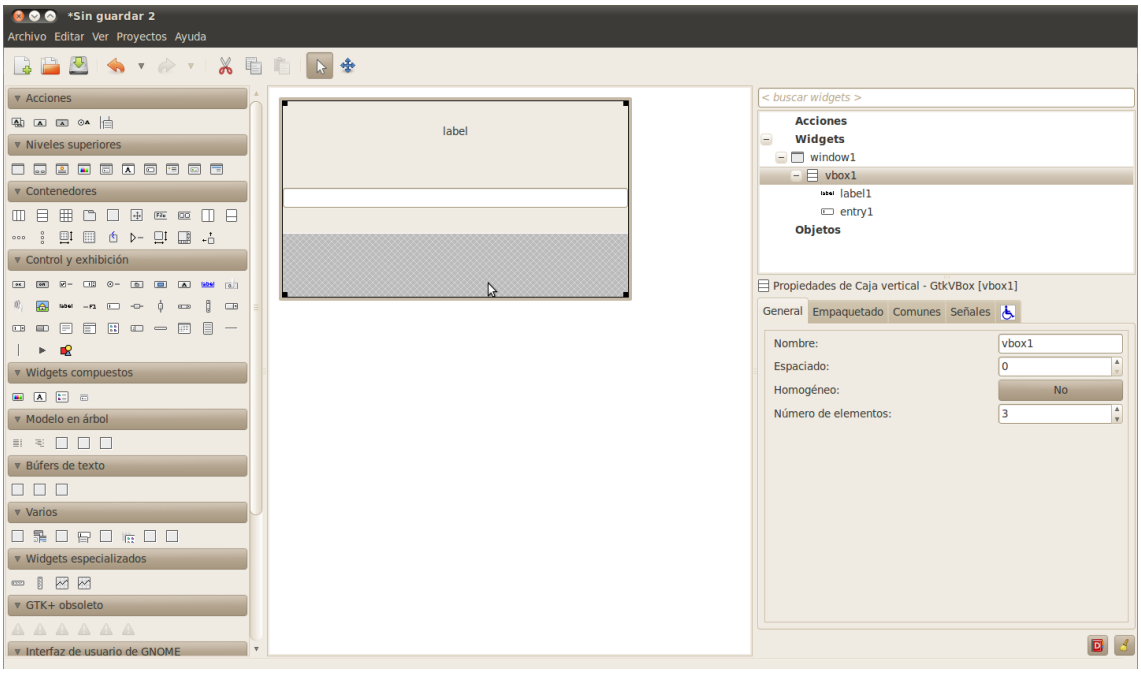
Anexo 4. Edición de propiedades de un widget:



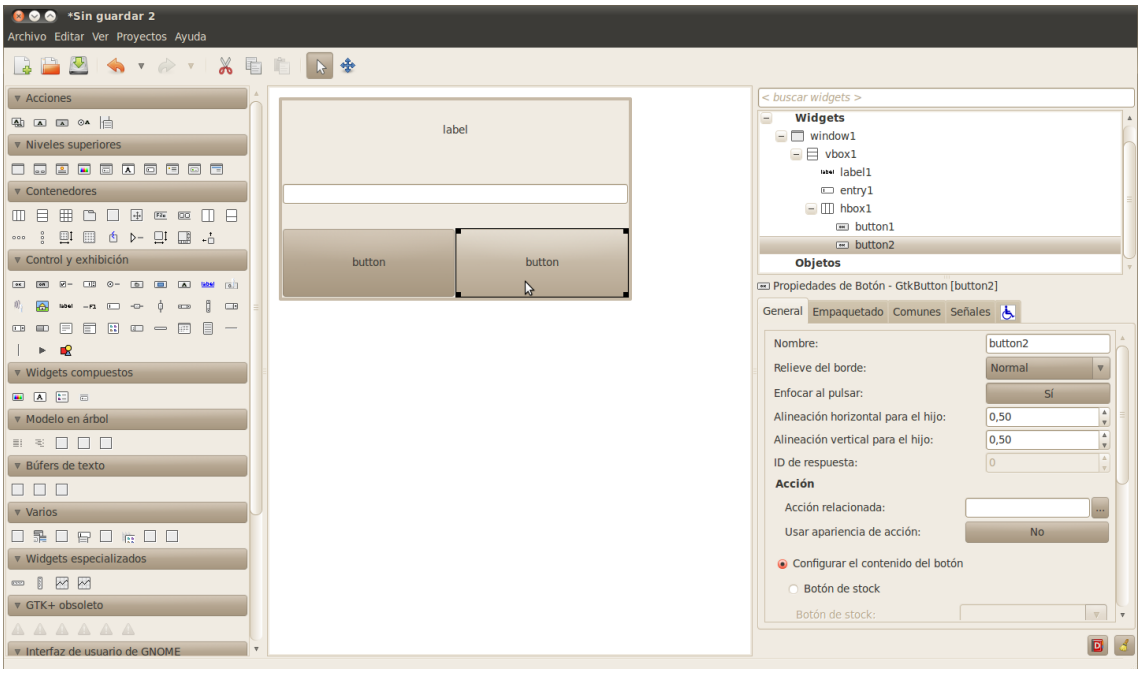
Anexo 5. Creación widget caja vertical:



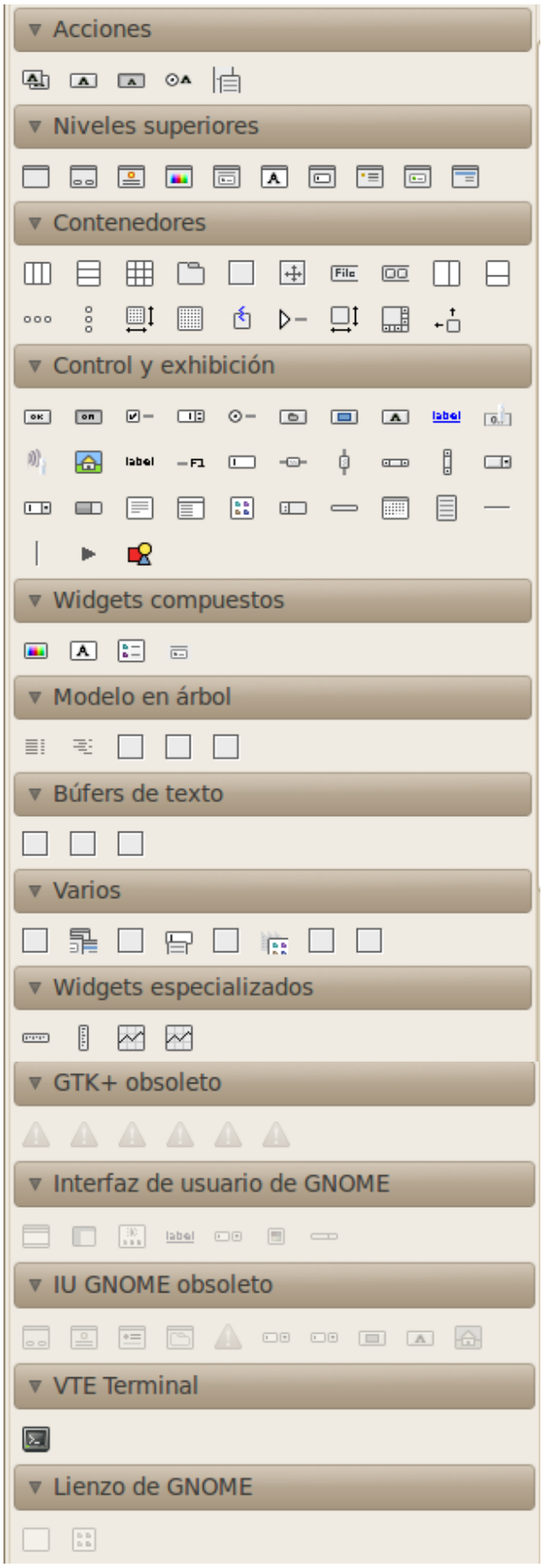
Anexo 6. Adición de widgets a un contenedor, entrada de texto y etiqueta:



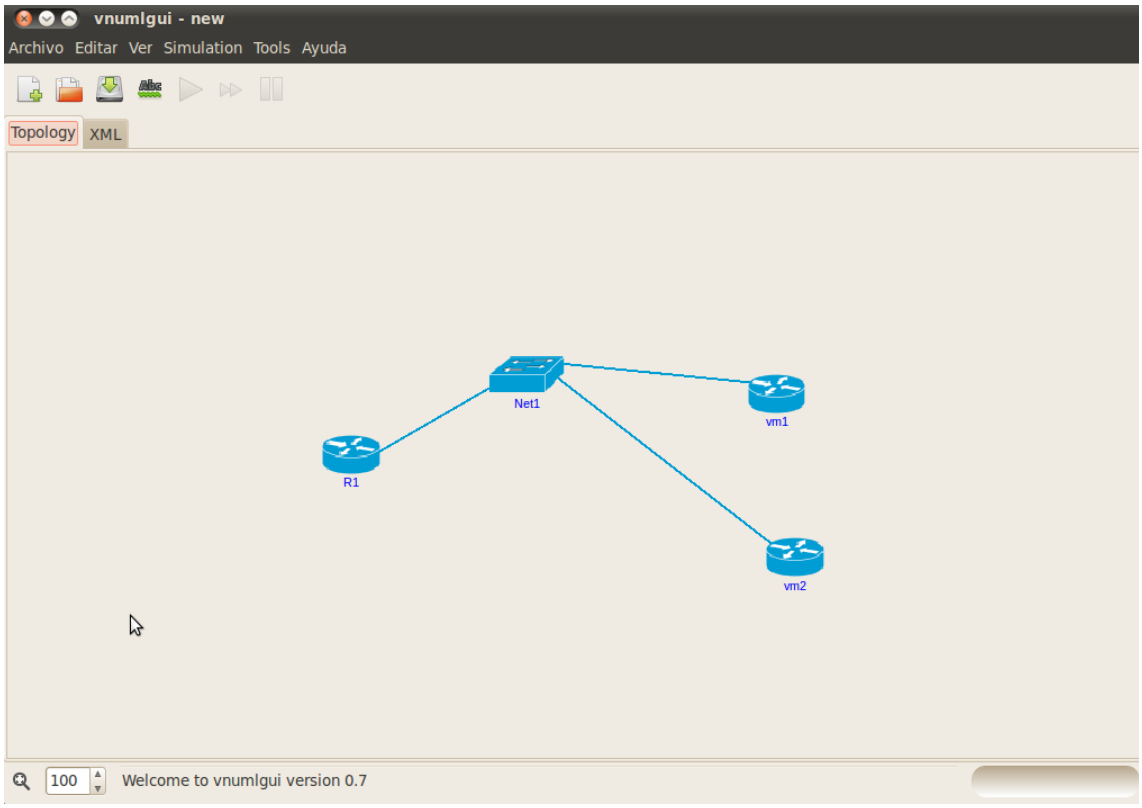
Anexo 7. Ejemplo de interfaz gráfica diseñada en glade:



Anexo 8. Widgets disponibles glade 3:



Anexo 9. Panel de diseño de topología de red de la interfaz gráfica VNUMLGUI:



Anexo 10. Generación del XML de la interfaz gráfica VNUMLGUI: