TECNOLOGÍA XML EN LA HERRAMIENTA DE SIMULACIÓN DE REDES VNUML

Fermín Galán Márquez[1]
Marzo 2004

Abstract

*Hoy en día, las técnicas de virtualización suponen una importante alternativa a la implementación de sistemas con equipos reales. El artículo analiza las motivaciones de la virtualización y presenta VNUML (*Virtual Network User Mode Linux*), una herramienta potente y flexible que simplifica el trabajo del usuario automatizando la construcción de escenarios virtuales. VNUML está basado en XML y estándares afines, analizando el artículo de que forma son utilizados.*

Palabras claves: virtualización, simulación, UML, XML.

1. Introducción

En la actualidad, la potencia del hardware disponible (equipos cada vez más rápidos en términos de CPU, con mayor cantidad de memoria RAM y espacio de almacenamiento en disco duro) ha motivado el interés por las técnicas de virtualización y sus aplicaciones, con el objetivo principal de reducir costes de despliegue y gestión de lo que sería un sistema equivalente realizado de forma convencional con equipos reales.

El presente artículo describe VNUML (*Virtual Network User Mode Linux*)[2], una herramienta basada en software libre cuyo objetivo es la construcción de entornos virtuales o simulaciones, desarrollada dentro del marco del proyecto Euro6IX [1] con la financiación parcial de la Comisión Europea. La motivación de esta herramienta es abstraer la simulación a un nivel descriptivo, ocultando los detalles complejos que un usuario no tendría por que conocer. Entre sus posibles aplicaciones se encuentran el desarrollo de maquetas de prueba, el despliegue de laboratorios, las redes señuelo o el *hosting*.

En primer lugar, se realizará una introducción a la virtualización, analizando sus principales ventajas e inconvenientes (sección 2) y exponiendo casos de aplicación orientados a VNUML (sección 3). A continuación, se describirá la herramienta VNUML (sección 4), describiendo como funciona, haciendo especial mención a como XML y estándares afines son utilizados (sección 5). Se incluye un ejemplo de uso (sección 6). Por último, se presentan las conclusiones del artículo (sección 7).

2. Virtualización

Desde un punto genérico, puede definirse la *virtualización* como una técnica (normalmente con un software asociado) que permite encapsular una unidad de proceso (ya sea un programa, un sistema operativo o incluso un equipo completo, dependiendo de a que

[1] Este trabajo está basado en el artículo "VNUML: Una herramienta de virtualización de redes basada en software libre" (Fermín Galán, David Fernández) publicado en la Conferencia Internacional de Software Libre de Málaga, Febrero 2004.
[2] La herramienta puede descargarse libremente desde http://www.dit.upm.es/~vnuml.

profundidad se sitúe el nivel de virtualización) para su ejecución dentro de un entorno en un equipo anfitrión que emula el entorno real transparentemente.

Esto implica que, en disposición de una máquina lo suficientemente potente que actúe de equipo anfitrión, es posible ejecutar simultáneamente un sistema de "máquinas virtuales" que se comporten de forma equivalente al mismo sistema implementado con máquinas reales (Figura 1). El grado de similitud entre la implementación virtual del sistema y la implementación con equipos reales puede tomarse como una medida de la bondad de la técnica de virtualización empleada: lo deseable es que el sistema emulado se comporte lo más transparentemente posible, idealmente exactamente igual que el sistema real.

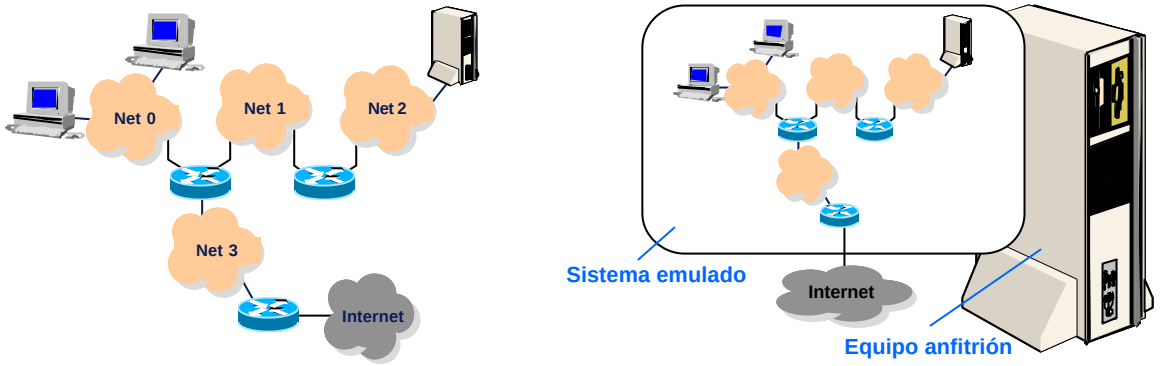

Figura 1. Sistema real (izquierda) y sistema virtual equivalente (derecha)

Las principales ventajas de la virtualización son el ahorro de costes de infraestructura y la simplificación de la gestión. En primer lugar, el utilizar un único equipo para implementar toda una infraestructura (formada por varios equipos, incluyendo posiblemente la infraestructura de red para interconectarlos) supone un ahorro de costes, tanto mayor cuando más complejo (más equipos) contiene el sistema original. A pesar de que, como veremos a continuación, el equipo anfitrión ha de ser suficientemente potente como para ejecutar entornos virtuales, el incremento de costes que esto puede suponer se compensa sobradamente en la mayor parte de los casos. En todo caso, normalmente es el coste el factor determinante a la hora de elegir una solución basada en virtualización.

La virtualización también facilita la gestión del sistema emulado. Un entorno formado por múltiples equipos interconectados es complejo de gestionar, ya que las acciones de gestión (configuración, corrección de errores, actualización de software, etc.) involucran varios elementos. En el caso de utilizar virtualización, hay que actuar en un único punto (el sistema anfitrión).

Sin embargo, la transparencia completa usando virtualización es difícilmente alcanzable. Esto es debido a que la virtualización introduce un nivel de proceso adicional (el que traduce las llamadas al sistema de la máquina virtual al sistema anfitrión) que supone un *overhead* debido al consumo de recursos. Para conseguir un rendimiento equivalente al del sistema real es necesario utilizar hardware de mayor potencia en la máquina anfitriona. En todo caso, esto es algo que depende de lo eficiente que sea la técnica de virtualización que se emplee. En la sección 4, se analizarán este consumo de recursos para el caso de VNUML/UML.

Si bien la teoría en la que se basan las técnicas de virtualización modernas es bastante antigua [2], el auge de este tipo de soluciones no se ha producido hasta hace relativamente poco tiempo, motivadas principalmente por la baja relación potencia del hardware/precio a la que

hemos llegado en nuestros días: Plex86 [3], Bochs [4], UML [5], VMware [6], … Como se ha señalado al principio de la sección, la virtualización puede producirse a distintos niveles: desde virtualización de procesos con aplicaciones de seguridad (como el mecanismo de *jails* en FreeBSD [7]) hasta emulación virtual de hardware (como en la que se basa VMware [6]). VNUML se basa en UML (*User Mode Linux* [5]), situado entre ambos extremos, que permite la emulación de sistemas operativos Linux.

3. Aplicaciones

VNUML está orientado a la prueba de aplicaciones de red y servicios sobre escenarios complejos compuestos de múltiples nodos (incluso decenas), evitando el coste de adquirir y gestionar equipos. Se señalan a continuación varios casos típicos en los que la herramienta puede ser de utilidad.

- Prueba de aplicaciones de red. Con VNUML es sencillo establecer una maqueta o demostrador para pruebas de aplicaciones de red (tales como servidores web, DNS, mail, etc.), especialmente cuando dichas pruebas están orientada a la funcionalidad de herramientas más que a medidas de eficiencia. Dentro el marco del proyecto Euro6IX [1], VNUML ha sido utilizado para evaluar y comparar distintas alternativas de software de *routing* BGP [8].

- Despliegue de laboratorios. Para una universidad o centro de enseñanza, el despliegue y mantenimiento de un laboratorio de ordenadores supone un coste importante. VNUML puede ayudar en la definición e implementación de dichos laboratorios utilizando técnicas de virtualización. En un caso extremo, el laboratorio entero podría ser incluido en CD-ROM (junto con algún programa de aprendizaje guiado) y proporcionado al alumno para su utilización independiente.

- Redes señuelo. Una red señuelo (*honeynet*) es una red cuyo objetivo es el estudio de amenazas de seguridad dirigidas contra ella [9]. Como no es una red de producción, no importa que los atacantes la dañen. Utilizando VNUML se simplifica la implantación y gestión de este tipo de redes (cuando la red ha sido comprometida, basta con recuperar un *backup* de los sistemas de fichero originales y regenerarla), siempre que se base en GNU/Linux. No obstante, cuando la red ha sido implementada utilizando técnicas de virtualización existen mecanismos mediante los cuales el atacante puede descubrir que se trata de una red virtual y no una real.

- *Hosting*. Para una empresa dedicada a la provisión de servicios de hospedaje (*hosting*) es posible utilizar un único equipo anfitrión sobre el que implementar un conjunto de servidores virtuales con VNUML, alquilándose su uso a los clientes. El cliente tiene la percepción de disponer de una máquina convencional, aunque realmente no sea así.

4. VNUML

VNUML es una herramienta software libre formada por dos componentes: un lenguaje simple y descriptivo basado en XML (*eXtended Markup Language* [10]) que permite al usuario definir el sistema emulado en un fichero de texto; y un parser (escrito en Perl [11]) de ese lenguaje que se encarga de procesar el fichero, construyendo la simulación y gestionándola, ocultando los detalles complejos al usuario (Figura 2). Esta sección describe

las características de VNUML, la sección 5 profundiza en su funcionamiento. La sección 6 muestra un ejemplo de uso de VNUML.

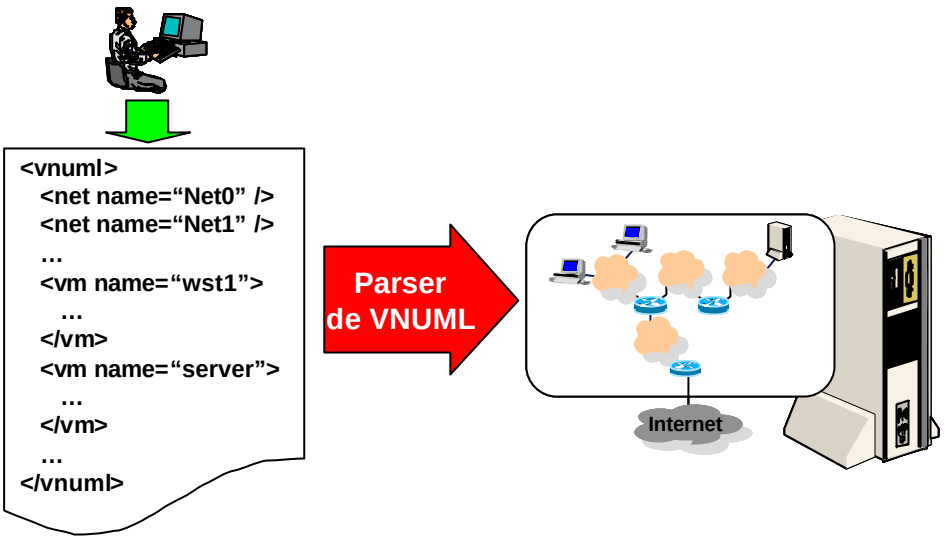

Figura 2. Funcionamiento básico de VNUML

XML es el lenguaje base para la descripción de escenarios de simulación. Propuesto por el W3C en 1998 tras el auge del lenguaje HTML (*Hypertext Markup Language*) en la web, XML surge como un estándar abierto para el intercambio de información por Internet basado en etiquetas de texto. Comparado con otras alternativas, esta orientación a etiquetas hace que el procesado del lenguaje no sea muy eficiente, pero sí muy sencillo y (como se puede comprobar en la Tabla 1), muy intuitivo para el usuario, características por las cuales se ha elegido en VNUML para representar las descripciones de escenario. XML tiene a su vez un conjunto de estándares asociados, de los que se hace uso en VNUML: DTD (*Document Type Definition*), para la definición de la estructura de etiquetas; y DOM (*Dinamic Object Model*), utilizado por el parser para procesar los datos de la simulación.

Como sistema de virtualización, VNUML utiliza UML. UML es una modificación de las fuentes del núcleo de Linux [12] que permiten su ejecución como proceso de usuario encima del núcleo convencional de Linux (el que ejecuta la máquina anfitriona). Cada uno de los procesos UML tiene asociados sus propios recursos (espacio de memoria, procesos, sistemas de ficheros, dispositivos de red, etc.) y, en definitiva, constituye una máquina virtual dentro de la cual otros procesos se ejecutan (Figura 3). La funcionalidad del núcleo UML es exactamente la misma que la de un núcleo convencional de Linux (de hecho, el proceso de compilación es análogo), por lo que la transparencia es (salvo por la pérdida de rendimiento debido al *overhead* de virtualización) total.

UML no solo proporciona la manera de crear máquinas virtuales, sino que existen mecanismos para la interconexión entre ellas mediante redes virtuales. Desde el punto de vista de la máquina virtual, se utiliza una interfaz de red de forma transparente. En el entorno anfitrión, la red virtual interconecta los distintos terminales utilizando el módulo de túneles (tun) del núcleo. Es de destacar como la interconexión se produce en modo *bridge*, a nivel 2, y, a todos los efectos, es como si las máquinas interconectadas a una misma red virtual estuvieran físicamente conectadas a un mismo segmento. También es posible que la máquina anfitriona intervenga en la simulación o interconectar máquinas virtuales con equipos externos a través del interfaz físico (lo cual, como se señalará en la sección 7, es importante a

la hora de integrar sistemas no-Linux en la simulación), de forma completamente transparente o a través de VLANs (*Virtual Local Area Network* [13]).

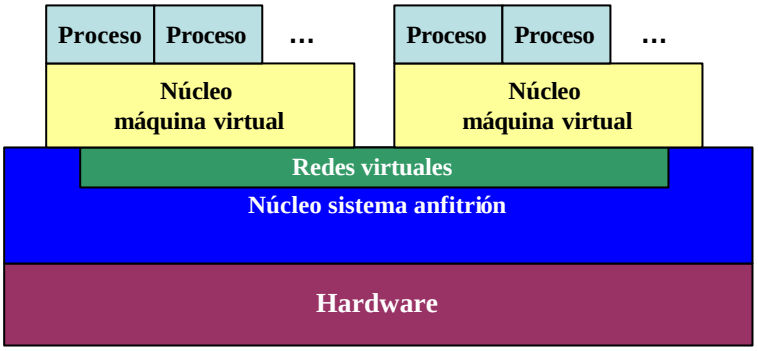

Figura 3. Virtualización con VNUML

UML es una herramienta potente pero compleja si se pretende construir escenarios que incluyan muchas máquinas virtuales y topologías complicadas de red. Además, es necesario tener un buen conocimiento de ciertos detalles del sistema Linux (dispositivos tap, *sockets* UNIX, *bridges* virtuales, etc.) para construir escenarios "a mano". La motivación con la que VNUML ha sido desarrollado es evitar esta complejidad, de forma que el usuario tenga que concentrarse solo en la definición del sistema emulado, como se verá en el ejemplo de la sección 6.

VNUML también proporciona facilidades en la gestión de las máquinas virtuales. En todas ellas existe una interfaz de red reservada, que permite una conexión directa con la máquina anfitriona, con el objetivo de realizar operaciones de gestión a través de la misma (por ejemplo, a través de SSH). Existen otros mecanismos de gestión de las máquinas virtuales, como el uso de consolas directas en las mismas (por ejemplo, un xterm) o a bajo nivel a través de un *socket* UNIX que comunica directamente con el núcleo UML.

El uso de VNUML/UML implica un consumo de recursos que ha de ser tenido en cuenta en el equipo anfitrión que albergará la simulación. En concreto, se analizan tres recursos: uso de CPU, memoria física y espacio de almacenamiento en disco.

- Uso de CPU. En general (aunque es algo que depende de los procesos que se ejecuten dentro de las máquinas virtuales), el consumo no es muy elevado. Como cifra, la CPU de un Pentium III 900 MHz utilizando en pruebas con más de veinte máquinas virtuales ejecutando software BGP [14] se encuentra ocupada tan solo al 10%. La excepción la constituye el momento de arrancar las máquinas virtuales, donde se producen picos elevados de consumo de CPU, que, en todo caso, es un proceso que solo hay que realizar una sola vez en toda la simulación. De todas formas, los desarrolladores de UML proporcionan mejoras (como Skas) orientadas al aumento en la eficiencia del uso de la CPU (hasta el cuádruple con respecto al modo convencional [15]).

- Memoria física. Normalmente es el factor limitante (a no ser que estén utilizando en las máquinas virtuales procesos intensivos en CPU). Cada máquina virtual tiene su propio espacio de memoria (de tamaño configurable), ubicado en el sistema de memoria virtual del núcleo de la máquina anfitriona. Ha de tenerse precaución de que el total de memoria asociado a las máquinas virtuales no provoque situaciones de

thrashing en la memoria virtual, lo que daría lugar a una severa disminución de la eficiencia.

- Almacenamiento en disco. Cada máquina virtual tiene asociado uno (o varios) sistemas de ficheros, que residen en un fichero (normalmente de gran tamaño) dentro del sistema de ficheros de la máquina anfitriona. En principio, el consumo de disco duro sería directamente proporcional al número de máquinas virtuales. Sin embargo, UML incorpora una optimización de escritura-en-copia (*Copy-on-write* o COW) con la cual varias máquinas virtuales comparten un mismo sistema de ficheros y solo se almacenan los cambios que cada una hace al mismo. De esta forma, se optimiza el espacio ocupado en disco duro.

VNUML se distribuye bajo los términos de la licencia GNU y ha seguido un modelo de desarrollo basado en software libre, donde cualquier usuario tiene acceso al código, puede realizar modificaciones sobre el mismo para adaptarlo a sus necesidades (como de hecho ha ocurrido) y puede contribuir al desarrollo del proyecto. La página web del proyecto [16] incluye el software para su descarga, así como documentación, tutoriales y soporte para los usuarios.

5. Funcionamiento

El procesado de los ficheros de descripción de escenario creados por el usuario se realiza mediante el parser de VNUML, invocado mediante un comando de consola en el equipo anfitrión. El programa trabaja en dos etapas (Figura 4):

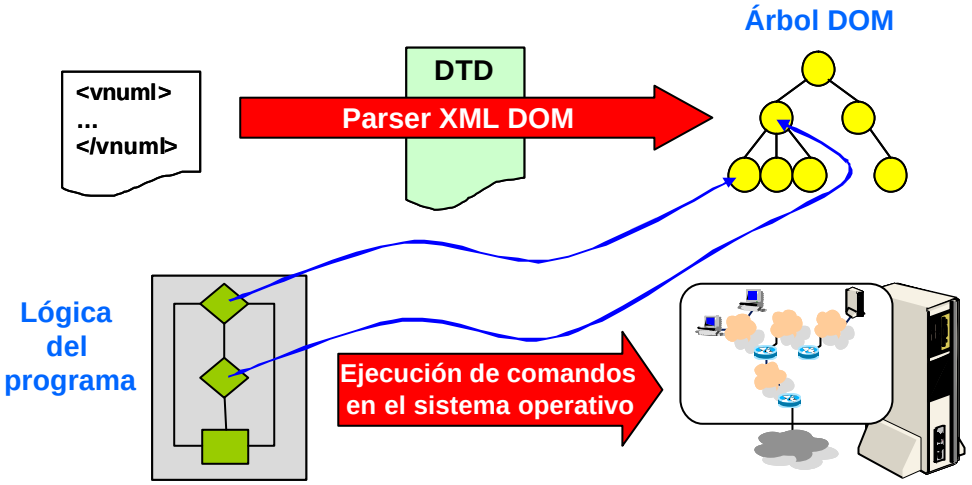

Figura 4. Funcionamiento detallado del parser de VNUML

- En primer lugar, un parser XML DOM (*Dinamic Object Model*) procesa el fichero VNUML, construyendo en la memoria del programa un árbol DOM que contiene toda la información del fichero XML. Es de destacar la utilización de un parser validador basado en DTD (*Document Type Definition*), de forma que si la especificación VNUML de entrada no es formalmente válida, el programa interrumpe su ejecución con un mensaje de error. La DTD se encuentra como referencia en el Anexo A del presente trabajo.

- Una vez el árbol DOM ha sido creado con los datos de la simulación, la lógica de control del programa se encarga de acceder a esos datos e invocar determinados comandos en el equipo anfitrión (relacionados con UML y la gestión de redes

virtuales mediante *bridges* virtuales), que conducen a la creación, gestión o eliminación de la simulación. Las acciones realizadas por esta lógica de control automáticamente serían las que tendría que realizar el usuario "a mano" de no utilizar VNUML.

En realizad, no existe una única lógica de control. El parser de VNUML utiliza cuatro modos de funcionamiento: construcción de topología (para crear el escenario en un sistema anfitrión en el que en principio no existe nada), parada y arranque de simulación (una manera de automatizar la ejecución de *scripts* en los nodos del sistema emulado, según la convención semántica que elija el propio usuario) y destrucción de topología (elimina el escenario, liberando los recursos, aunque el contenido del sistema de ficheros de las máquinas virtuales es persistente). El procesado de un mismo fichero VNUML es distinto en cada caso[3].

Es de destacar el uso de DOM para el procesado de XML frente a alternativas que procesan el XML de forma secuencial (como SAX). Esto es debido a que la lógica de control que procesa los datos necesita acceder a ellos de forma arbitraría, por lo que la aproximación secuencial al procesado no es viable.

Si bien se realiza una validación basada en DTD, es necesario un nivel adicional de validación implementado dentro de la lógica de control. La DTD permite validar fácilmente la estructura del XML, pero presenta carencias semánticas al comprobar el formato de los datos. Por ejemplo, no es posible comprobar si el valor de una etiqueta es una dirección IPv4 o IPv6 válida. Para suplir esta carencia podría utilizarse un mecanismo de validación más potente (tipado de datos más fuerte), como XML Schema [20]. En cualquier caso, existen comprobaciones que van más allá del formato de los datos (como por ejemplo, comprobar que los ficheros identificados por ciertos valores de etiqueta existen realmente) y que requieren de validación implementada por el propio programa.

El parser de VNUML ha sido íntegramente escrito en Perl. Originalmente concebido como un lenguaje aglutinador especializado en el procesamiento de expresiones de texto y en *scripts* para servidores web, hoy en día se ha convertido en un lenguaje de propósito general de gran auge. Se trata de un lenguaje de programación moderno (modular, orientado a objetos) interpretado para el que existen intérpretes en múltiples plataformas (GNU/Linux, Solaris, Windows, etc.). Se ha elegido Perl por la orientación de este lenguaje al procesado de expresiones de texto y por la facilidad de mantenimiento del programa (depuración, modificación, etc.) que aporta un lenguaje interpretado de este tipo. En concreto, los módulos XML::DOM y XML::DOM::Valparser han sido utilizados para procesar XML.

6. Ejemplo de uso

A continuación se incluye un ejemplo simplificado de aplicación de VNUML. Considérese que se quiere emular una topología como la que muestra la Figura 5. El fichero de configuración VNUML que define el anterior sistema es el que se muestra en la Tabla 1.

Como puede observarse, el fichero de configuración es bastante sencillo: no es necesario que el usuario conozca detalles complejos de UML o Linux para utilizar la herramienta. Cada una de las máquinas virtuales está definida dentro de una etiqueta *vm* (existe una etiqueta especial, *host*, que define el entorno de la máquina anfitriona). Dentro de ella, la etiqueta

[3] Realmente, en los modos de arranque y parada de simulación el procesamiento es casi el mismo.

filesystem indica cual es el fichero que alberga el sistema de ficheros raíz que utilizará esa máquina virtual.

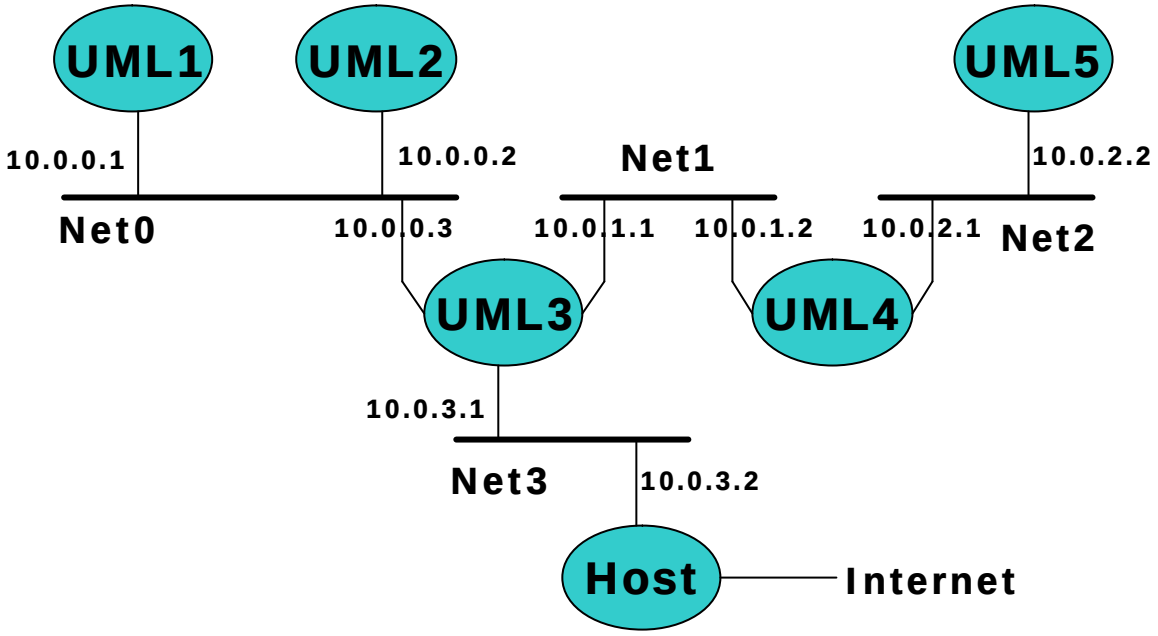

Figura 5. Ejemplo de escenario VNUML

La etiqueta *if* define una interfaz de red en la máquina virtual. Observar como una misma máquina (por ejemplo, UML3 y UML4) puede definir varias interfaces. La interconexión se realiza definiendo redes virtuales con la etiqueta *net*. El nombre de estas redes es el que se utiliza como atributo de *if* para interconectar una interfaz a una determina red. Observar como también se configuran las rutas que permitirán la interconexión (*route*) y el reenvío de paquetes (*forwarding*).

El ejemplo se ha escrito utilizando IPv4. También es posible utilizar IPv6 (en ese caso, se utilizaría la etiqueta *ipv6* en vez de *ipv4* para asignar direcciones).

Tabla 1. Descripción VNUML de la simulación

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM
  "/var/vnuml/vnuml.dtd">
<vnuml>
  <net name="Net0"/>
  <net name="Net1"/>
  <net name="Net2"/>
  <net name="Net3"/>
  <vm name="uml1">
    <filesystem>/var/vnuml/um1 1.fs</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.0.0.1</ipv4>
    </if>
    <route type="inet" gw="10.0.0.3">default</route>
  </vm>
  <vm name="uml2">
    <filesystem>/var/vnuml/uml2.fs</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.0.0.2</ipv4>
    </if>
     <route type="inet" gw="10.0.0.3">default</route>
     <start>/usr/local/samba/smbd</start>
     <stop>killall smb</stop>
  </vm>
  <vm name="uml3">
    <filesystem>/var/vnuml/uml3.fs</filesystem>
    <if id="1" net="Net0">
      <ipv4>10.0.0.3</ipv4>
    </if>
    <if id="2" net="Net1">
      <ipv4>10.0.1.1</ipv4>
    </if>
    <if id="3" net="Net3">
      <ipv4>10.0.3.1</ipv4>
    </if>
    <route gw="10.0.1.2">10.0.2.0/24</route>
    <route gw="10.0.3.2">default</route>
    <forwarding />
  </vm>
  <vm name="uml4">
    <filesystem>/var/vnuml/uml4.fs</filesystem>
    <if id="1" net="Net1">
      <ipv4>10.0.1.2</ipv4>
    </if>
    <if id="2" net="Net2">
      <ipv4>10.0.2.1</ipv4>
    </if>
    <route gw="10.0.1.1">default</route>
    <forwarding />
  </vm>
  <vm name="uml5">
    <filesystem>/var/vnuml/uml5.fs</filesystem>
    <if id="1" net="Net2">
      <ipv4>10.0.2.2</ipv4>
    </if>
    <route gw="10.0.2.1">default</route>
  </vm>
  <host>
    <hostif net="Net3">
      <ipv4>10.0.3.2</ipv4>
    </hostif>
     <route gw="10.0.3.1">10.0.0.0/16</route>
     <forwarding />
  </host>
</vnuml>
```

Una vez escrito el fichero (por ejemplo, *ejemplo.xml*), se invoca al parser con un comando de consola en modo construcción de topología para que lo interprete y cree el escenario.

vnuml –t ejemplo.xml

Una vez el parser acaba su proceso, la simulación ya está en ejecución y puede interaccionarse con ella exactamente igual que si fuera un sistema real.

Los comandos pueden ser ejecutados en las máquinas virtuales mediante SSH o consola, si bien VNUML permite la definición de secuencias de comandos para su ejecución automática dentro de ciclos de simulación. Esto se realiza con las etiquetas *start* y *stop*, para la ejecución de comandos al inicio y finalización de la simulación (corresponde al usuario definir la semántica de este inicio y finalización). Por ejemplo, en la máquina UML2 al inicio de simulación se arranca un servidor de ficheros Samba [17] (*/usr/local/samba/smbd*) y al finalizar se para (con una señal de terminación mediante killall).

En el ejemplo anterior, cada máquina virtual utiliza un *filesystem* distinto. Suponiendo que todos ellos tengan un tamaño similar de, por ejemplo, 300 Mbytes, el consumo total de disco duro sería 1.500 Mbytes. Es posible utilizar COW para reducir el consumo al de un solo sistema de ficheros (300 Mb), prácticamente. Sustituyendo las etiquetas como muestra la Tabla 2, todas las máquinas virtuales utilizan al arrancar el mismo *filesystem*, guardándose solo las modificaciones (que, normalmente, tienen un tamaño despreciable frente al del filesystem).

Tabla 2. Filesystems COW

```
…
<filesystem type="cow">/var/vnuml/uml.fs</filesystem>
…
<filesystem type="cow">/var/vnuml/uml.fs</filesystem>
…
<filesystem type="cow">/var/vnuml/uml.fs</filesystem>
…
<filesystem type="cow">/var/vnuml/uml.fs</filesystem>
…
<filesystem type="cow">/var/vnuml/uml.fs</filesystem>
…
```

7. Conclusiones

Las técnicas de virtualización, utilizadas de forma conveniente, pueden suponer un importante ahorro de costes en la implementación de un sistema. Por ejemplo, considerando el caso de la Figura 5, utilizar una única máquina para implementar el sistema reduciría los costes al 20% (considerando únicamente las máquinas; el ahorro sería mayor si consideramos también la eliminación de cables, *hubs* y otros elementos de interconexión). En realidad, es posible que el equipo anfitrión necesite más potencia que un equipo convencional y debido a ello resulte más caro, pero, teniendo en cuenta que los recursos más críticos (RAM y espacio en disco duro) son relativamente baratos hoy en día, el margen de ahorro seguirá siendo considerable.

VNUML es una herramienta de virtualización de propósito general flexible y que simplifica el trabajo del usuario ofreciéndole gran parte de la potencia de UML pero sin tener que enfrentarse a los detalles complejos de creación de máquinas y redes virtuales. De esta forma, el usuario puede concentrarse en la definición de la simulación, que es lo realmente importante desde su punto de vista.

Una de las principales limitaciones de VNUML/UML es que solo permite la creación de máquinas virtuales Linux, lo cual puede suponer un problema si en el entorno a emular hay sistemas de otro tipo (por ejemplo, un router de Cisco o una base de datos Microsoft SQL Server). En tales casos, caben dos posibilidades: buscar una alternativa basada en GNU/Linux (para el caso anterior, podría utilizarse zebra [18] y MySQL [19], respectivamente) o construir una simulación en la que los sistemas no-Linux sean elementos externos. Una solución a más largo plazo, sería cambiar el *backend* de virtualización, sustituyendo UML por otro sistema que sea más flexible con respecto al tipo de sistemas que puede emular (como por ejemplo, VMware [6], que, entre otros, permite la emulación de máquinas Windows).

Los únicos casos en los que VNUML es claramente no utilizable son aquellos en los que se realicen medidas de eficiencia de aplicaciones, ya que el *overhead* introducido por la virtualización falsearía estas medidas.

Otras líneas de trabajo que se contemplan son la posibilidad de introducir estímulos externos en la simulación (por ejemplo, la caída de un enlace de red en un determinado instante de tiempo) o la vinculación entre VNUML y otras herramientas de simulación como ns [21] (simulador de red basado en eventos discretos), mediante la construcción de traductores entre lenguajes de especificación.

Con respecto a la utilización de XML y tecnologías relacionadas (DTD y DOM), es de destacar la flexibilidad de este lenguaje (en sus orígenes creado para el intercambio de información entre máquinas a través de Internet), demostrándose en VNUML su gran versatilidad en aplicaciones en las que es necesario el estructurado de datos para su posterior procesado de forma sencilla. En este campo, pueden considerarse como líneas de trabajo la utilización de mecanismos de validación más potentes basados en XML Schema (como se introduce en la sección 5) o la creación de un *front-end* gráfico que facilite la creación y edición de descripciones VNUML (XML es un lenguaje tedioso para ser utilizado por humanos).

Anexo A: DTD de VNUML

```
<!-- VNUML DTD version 1.3 -->
<!ELEMENT vnuml (global,net*,vm*,host?)>
<!ELEMENT global
(version,simulation_name,ssh_key?,automac?,ip_offset?,netconfig?,host_mapping?,shell?,tun_device?)>
<!ELEMENT net EMPTY>
<!ATTLIST net name CDATA #REQUIRED
          external CDATA #IMPLIED
          vlan CDATA #IMPLIED>
<!ELEMENT vm (filesystem,mem?,kernel?,boot?,if*,route*,forwarding?,filetree*,start*,stop*)>
<!ATTLIST vm name CDATA #REQUIRED>
<!ELEMENT host (hostif*,physicalif*,route*,forwarding?,start*,stop*)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT simulation_name (#PCDATA)>
<!ELEMENT ssh_key (#PCDATA)>
<!ELEMENT automac EMPTY>
<!ATTLIST automac offset CDATA "0">
<!ELEMENT ip_offset (#PCDATA)>
<!ELEMENT netconfig EMPTY>
<!ATTLIST netconfig stp (on|off) "off"
                promisc (on|off) "on">
<!ELEMENT host_mapping EMPTY>
<!ELEMENT shell (#PCDATA)>
<!ELEMENT tun_device (#PCDATA)>
<!ELEMENT filesystem (#PCDATA)>
<!ATTLIST filesystem type (direct|copy|cow) #REQUIRED>
<!ELEMENT mem (#PCDATA)>
<!ELEMENT kernel (#PCDATA)>
<!ELEMENT boot (con0?)>
<!ELEMENT if (mac?,ipv4*,ipv6*)>
<!ATTLIST if id CDATA #REQUIRED
         net CDATA #REQUIRED>
<!ELEMENT route (#PCDATA)>
<!ATTLIST route type (inet|inet6) #REQUIRED
           gw CDATA #REQUIRED>
```

```
<!ELEMENT forwarding EMPTY>
<!ATTLIST forwarding type (ip|ipv4|ipv6) "ip">
<!ELEMENT filetree (#PCDATA)>
<!ATTLIST filetree root CDATA #REQUIRED
          when (start|stop|always) #REQUIRED>
<!ELEMENT start (#PCDATA)>
<!ATTLIST start type (verbatim|file) #REQUIRED>
<!ELEMENT stop (#PCDATA)>
<!ATTLIST stop type (verbatim|file) #REQUIRED>
<!ELEMENT hostif (ipv4*,ipv6*)>
<!ATTLIST hostif net CDATA #REQUIRED>
<!ELEMENT con0 (#PCDATA)>
<!ELEMENT mac (#PCDATA)>
<!ELEMENT ipv4 (#PCDATA)>
<!ATTLIST ipv4 mask CDATA "255.255.255.0">
<!ELEMENT ipv6 (#PCDATA)>
<!ELEMENT physicalif (#PCDATA)>
<!ATTLIST physicalif name CDATA #REQUIRED
          ip CDATA #REQUIRED
          mask CDATA #REQUIRED
          gw CDATA #REQUIRED>
```

Referencias

[1] "Euro6IX", http://www.euro6ix.org/, tomado el 8 de Marzo de 2004.

[2] L. Seawright y R. MacKinnon, "VM/370: A Study of Multiplicity and Usefulness", *IBM Systems Journal*, 18(1), 1979.

[3] "Plex/86", http://plex86.sourceforge.net/, tomado el 8 de Marzo de 2004.

[4] "Bochs IA-32 Emulator", http://bochs.sourceforge.net/, tomado el 8 de Marzo de 2004.

[5] J. Dike, "User Mode Linux", *Proc. 5th Annual Linux Showcase & Conf.*, Oakland CA, 2001.

[6] J. Nieh y O. C. Leonard, "Examining VMware", *Dr. Dobb's Journal*, Agosto 2002.

[7] P. Hope, "Using Jails in FreeBSD for Fun and Profit", *Login: The Magazine of Usenix & Sage*, n. 3, vol. 27, Junio 2002.

[8] D. Fernández, F. Galán y T. de Miguel. "Study and Emulation of IPv6 Internet Exchange (IX) based Addressing Models". IEEE Communications Magazine, vol. 42, no. 1, Enero 2004

[9] E. Gallego y J. E. Lopez "Honeynets: aprendiendo del atacante", IX Congreso Nacional de Internet, Telecomunicaciones y Movilidad, Febrero 2004 (pendiente de publicación).

[10] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, Octubre 2000.

[11] "The Perl Directory", http://www.perl.org/, tomado el 8 de Marzo de 2004.

[12] "The Linux Kernel Archives", http://www.kernel.org/, tomado el 8 de Marzo de 2004.

[13] “802.1q: Virtual LANs”, IEEE 802.1Q Working Group, IEEE, 2001.

[14] Y. Rekhter y T. Li, “A Border Gateway Protocol 4 (BGP-4)”, RFC 1771, IETF, Marzo 1995.

[15] http://user-mode-linux.sourceforge.net/skas.html, tomado el 8 de Marzo de 2004.

[16] “VNUML Project Home Page”, http://www.dit.upm.es/~vnuml/, tomado el 8 de Marzo de 2004.

[17] “Samba”, http://www.samba.org/, tomado el 8 de Marzo de 2004.

[18] “GNU Zebra”, http://www.zebra.org/, tomado el 8 de Marzo de 2004.

[19] “MySQL”, http://www.mysql.com/, tomado el 8 de Marzo de 2004.

[21] “W3C XML Schema”, www.w3.org/XML/Schema, tomado el 8 de Marzo de 2004.

[21] “The Network Simulator”, http://www.isi.edu/nsnam/ns/, tomado el 8 de Marzo de 2004.