Seminar User Mode Linux
VNUML - Versionen 1.5 und 1.6

Wintersemester 2005/2006

Thomas Chmielowiec
Matrikelnummer: 203110481
Studiengang: Computervisualistik

Dozent: Dipl.-Inform. Harald Dickel

26. Februar 2006

Inhaltsverzeichnis

1 Einleitung

Nachdem bereits ein Einblick in die Arbeit mit VNUML gegeben wurde, ist es Ziel dieser Seminararbeit, tiefer in die Konfigurationsmöglichkeiten eines VNUML-Szenarios einzusteigen. Hierbei wird der Workflow vom Design bis zur tatsächlichen Umsetzung verfolgt. Der erste Teil befasst sich mit der Planung eines Szenarios und leitet mit ersten allgemeinen Informationen über die Struktur einer VNUML-Quellcode-Datei, über in die Beschreibung der einzelnen möglichen Tags. Diese Tags werden im Detail erklärt, und sind erstmal für sowohl Version 1.5, als auch für Version 1.6 gültig. Im dritten Teil werden die Unterschiede und Neuheiten in Version 1.6 herausgearbeitet. Letztendlich bietet Version 1.6 eine grundlegend andere Sicht auf das Starten und Modellieren eines Szenarios, wenn man die besondere Rolle von User- und Rootrechten unter Linux berücksichtigt.

2 Planung

Ein typischer Entwicklungsweg besteht aus folgenden Phasen:

Designphase Der Benutzer muss sich zunächst mal ein Simulationsszenario überlegen. Je nachdem was er testen und simulieren möchte, braucht er möglicherweise mehrere verschiedene virtuelle Maschinen die beteiligt sind und er muss sich eine Netzwerktopologie überlegen in der die virtuellen Maschinen miteinander kommunizieren. Dazu gehört auch, dass er weiß, welche Netzwerkinterfaces eine Maschine hat, wie sie konfiguriert sind, und wie genau sie miteinander vernetzt sind.

VNUML bietet auch die Möglichkeit virtuelle Maschinen Befehle ausführen zu lassen. Auch diese „Skripte“ müssen vorbereitet werden, dies wird aber Thema eines späteren Kapitels.

Nicht zu vergessen ist, dass die komplette Simulation auf dem Host, also auf derselben physikalischen Maschine läuft. So liegt es Nahe, dass auch der Host Teil der Simulation sein kann.

Implementationsphase Sobald das Design des Szenarios steht, muss es in “maschinenlesbarer“ Form in einer XML-Datei umgesetzt werden. So ein XML-Dokument ist idealerweise auch menschenlesbar und hat die Form einer Baumstruktur bestehend aus Elementknoten, Elementattributen und Elementinhalten. Diese Elementknoten sind in VNUML in 3 Typen aufgeteilt:

structural tags Sie definieren die Struktur der Datei, teilen sie in Sektionen bzw. Sinngruppen, und enthalten selbst auch wieder spezifische Tags. Sie haben wenig, oder gar keine Semantik.

topology tags Mit den topology tags, lässt sich das Szenario beschreiben. Diese Tags werden beim Starten und beim Beenden eines Szenarios ausgewertet und umgesetzt.

simulation tags Nachdem ein Szenario gestartet wurde, kann es in den executing commands mode gesetzt werden, um auf virtuellen Maschinen Befehle auszuführen. In den simulation tags werden diese Befehle vorher festgelegt.

3 Aufbau einer VNUML-Datei

Wie schon erwähnt, ist VNUML eine XML-Sprache. Wie in jeder XML-Datei sollte man eine DTD angeben, sozusagen ein Regelwerk, das dabei hilft die syntaktische Korrektheit der Datei festzustellen. Die ersten zwei Zeilen der Quelldatei sollten also wie folgt aussehen:

```
<?xml version="1.0"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
```

Dabei bezieht sich der Pfad der DTD-Datei auf eine Datei im Host-Filesystem. Kommentare in einer VNUML-Datei werden wie in HTML wie folgt eingefügt:

```
<!-- das ist ein Kommentar -->
```

Eine VNUML-Datei ist in vier große Abschnitte aufgeteilt, die durch vier structural tags gekennzeichnet werden:

1. `<global>`: Beschreibt globale Elemente der Simulation
2. `<net>`: Beschreibt die virtuellen Netze
3. `<vm>`: Spezifikation der virtuellen Maschinen
4. `<host>`: Konfiguration des Hosts

Wenn wir nun das komplette Grundgerüst unserer VNUML-Datei betrachten, so kommen wir zu folgendem Aufbau:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">
<vnuml>
  <global>
<!-- topology tags -->
  </global>
  <net name="Net0"/>
  ....
  <net name="Netx"/>
  <vm name="uml1">
<!-- topology tags -->
  </vm>
  ....
  <vm name="umlx">
<!-- topology tags -->
  </vm>
  <host>
<!-- topology tags -->
  </host>
</vnuml>
```

Wie im Beispiel zu sehen, enthalten die structural tags natürlich auch noch Untertags (mit der Ausnahme von `<net>`).Welche das sind, wie sie funktionieren, und welche Attribute man verwenden darf, bildet einen der zentralen Punkte dieser Seminarausarbeitung.

4 VNUML 1.5 - Die Tags im Detail

4.1 Globale Definitionen mit <global>

Die globalen Definitionen bilden die Rahmenbedingungen für die gesamte Simulation. Sie haben also Auswirkungen auf alle virtuellen Maschinen. Folgende wichtige Untertags sind verfügbar:

<version> Über dieses Tag gibt man die verwendete VNUML Version an.

<simulation_name> Dem Szenario muss ein eindeutiger Name zugewiesen werden. Das hat den Hintergrund, dass es möglich ist, mehrere Szenarios parallel laufen zu lassen. Der Name des Szenarios sollte auch dem Dateinamen der VNUML-Datei entsprechen. In diesem Falle würde die Datei also test.xml heißen.

<**ssh_key**> Hier kann der absolute Pfad der Datei mit dem SSH-Schlüssel des Hosts angegeben werden. Setzt man diesen optionalen Tag auf einen gültigen Wert, so wird der Schlüssel auf den virtuellen Maschinen installiert und es entfällt die Eingabe des SSH-Passworts. Vorher muss man über ssh-keygen einen public key erzeugen.

<**automac**> Jedes Interface der virtuellen Maschinen muss eine MAC-Adresse haben. Möchte oder braucht man diese nicht von Hand zu setzen, so kann man sich über dieses Tag MAC-Adressen für die Rechner generieren lassen. Über ein `offset`-Attribut kann man das Setzen der MAC-Adressen beeinflussen, um eine einwandfreie Zuordnung bei mehreren parallel laufenden Simulationen zu gewährleisten. Generiert werden die MAC-Adressen nach einem Schema, das abhängig vom Interface, der VM und des `offset`-Attributs ist. Selbst wenn man `<automac>` verwendet, hat die manuelle Angabe einer MAC-Adresse immer Vorang.

<**ip_offset**> Die virtuellen Rechner besitzen sogenannte Management interfaces (siehe ersten Seminarvortrag). Über dieses optionale Tag kann nun ein Offset definiert werden, der auf die IP-Adressen der Management Interfaces addiert wird. Man kann außerdem das Attribut `prefix` angeben, über das man die ersten zwei Byte der IP-Adressen der virtuellen Rechner verändern kann. Standardmäßig ist dies `192.168`.

<**host_mapping**> Ermöglicht das Mappen der IPs der virtuellen Rechner auf deren Namen, den sie im `<vm>`-Tag bekommen haben. So sind die Maschinen also nicht nur über ihre IPs, sondern auch über ihre Namen ansprechbar. Dies geschieht durch automatisches editieren der `/etc/hosts`-Datei auf dem Host. Diese wird beim Beenden des Szenarios wiederhergestellt.

<**shell**> Ein optionales Tag, das die Angabe eines Command Interpreters ermöglicht, der zum Ausführen der Kommandos des VNUML-Parsers verwendet werden soll. Die Standardshell ist `/bin/bash`.

<**default_filesystem**> Hier wird ein Pfad zum Dateisystem angegeben, das in der virtuellen Maschine verwendet werden soll. Das Dateisystem kann man auch explizit durch `<filesystem>` in der Definition der VM angeben, diese Angabe definiert aber den globalen Standard. Im Dateisystem müssen sich unter anderem Startskripte für SSH befinden, aber auch Programme wie ifconfig, echo und halt. Standardmäßig wird ein Dateisystem vewendet, dass bei der VNUML-Installation mitkommt.

<**default_kernel**> Wird bei der Definition der VM kein Kernel angegeben der auf der VM gebootet werden soll, so wird dieser auf den Standardkernel (`kernels/linux` des VNUML-Installationsverzeichnisses) gesetzt.

<**basedir**> Hier kann man einen offset-Pfad für den Tag `<filetree>` einer VM angeben. Ein Beispiel dazu wird bei der Definition einer VM gezeigt.

Ein Muster einer `<global>`-Sektion könnte also so aussehen:

```
<global>
    <version>1.5</version>
    <simulation_name>tutorial</simulation_name>
    <ssh_key>/root/.ssh/identity.pub</ssh_key>
    <automac/>
    <ip_offset>100</ip_offset>
    <host_mapping/>
</global>
```

4.2 Virtuelle Netze mit <net>

Das structural tag `<net>` hat keine Untertags. Es hat lediglich Attribute die dabei helfen, ein virtuelles Netz mit einem Namen einzurichten. Jedes mit `<net>` erstellte Netzwerk, ist irgendwie mit einem Interfaces des Hosts oder einer virtuellen Maschine verbunden. Folgende Attribute können oder müssen gesetzt werden:

- **name**: Identifiziert wird ein Netz anhand seines `name`-Attributes. Dieser Name wird bei der Definition der VMs zur Kopplung der Interfaces an das Netz verwendet. Diese Angabe ist Pflicht.

- **mode**: Über das Attribut `mode` kann man den Modus der Verbindung zwischen den virtuellen Rechnern setzen:

  mode="virtual_bridge" Es wird eine virtuelle Brücke zum Verbinden der Rechner verwendet. Über die Angabe von `external` lässt sich eine virtuelle Schnittstelle mit einer echten Schnittstelle des Hosts verbinden, und so eine Verbindung zum Internet realisieren. Ein Beispiel dazu wird bei in Kapitel 5.3 vorgestellt.

  mode="uml_switch" Im Hintergrund läuft ein uml_switch-Prozess der die virtuellen Netzwerke implementiert und verwaltet. Möchte man einen Hub anstatt eines Switches verwenden, nimmt man zusätzlich die Angabe `hub="yes"`. Das für den uml_switch-Prozess

nur Userrechte notwendig sind (für die virtuellen Brücken braucht man Rootrechte), wird in Kapitel 5 interessant.

Der Standardtyp ist "virtual bridge".

- **type**: Definiert den Typ des Netzwerkes. Mögliche Typen sind:

  lan emuliert ein konventionelles broadcast ethernet network

  ppp zwei virtuelle Maschinen werden über einen ppp-Link verbunden, über den Tag `<bw>` muss dann aber auch noch eine Bandbreite für den Link festgelegt werden.

Hier ein einfaches Standard-Beispiel:

```
<net name="Net0"/>
<net name="Net1"/>
<net name="Net2"/>
```

Die Netze erlauben es den virtuellen Maschinen also Daten untereinander auszutauschen. Es besteht zusätzlich noch die Möglichkeit so ein virtuelles Interface mit einem physikalischen Interface des Hosts zu verbinden, wenn man dem Netz noch ein `external`-Attribut gibt. So wird ermöglicht, dass virtuelle Maschinen Zugriff auf externe Netzwerke bekommen.

4.3 Virtuelle Maschinen mit <vm>

Jedes `<vm>`-Tag definiert eine eigene virtuelle Maschine. Um sie vollständig zu definieren, stehen eine Reihe von Untertags zur Verfügung. Zunächst gibt es zwei Attribute für das `<vm>`-Tag:

- **name:** Der Name der VM muss eindeutig sein. Die Länge des Namens ist auf 7 Zeichen beschränkt, und somit auch die theoretische maximale Anzahl der VMs. Denn je mehr VMs man simulieren möchte, desto mehr Resourcen werden auf dem Host benötigt.

- **order:** Über dieses Attribut können die virtuellen Maschinen höher oder niedriger priorisiert werden. Man kann damit z.B. beeinflussen welche Maschine zuerst gebootet/heruntergefahren wird. Standardmäßig haben alle VMs dieselbe Priorität, einzig die Definitionsreihenfolge in der VNUML-Datei ist noch ausschlaggebend.

Die Tags die in der `<vm>`-Umgebung verwendet werden dürfen, konfigurieren die virtuelle Maschine:

<**filesystem**> Dieses Tag ist optional, der Standardwert wird über `<default_filesystem>` ermittelt. Jeder UML-Kernel der gebootet wird, braucht solch ein Dateisystem. Bei normalen Linuxmaschinen liegt dieses Dateisystem normalerweise auf der root-Partition, hier wird eine Datei angegeben, in der das Dateisystem liegt. Der absolute Dateiname zur Filesystem-Datei wird innerhalb des `<filesystem>`-Tags angegeben.

Der Tag hat auch noch ein Attribut `type`. Dieses Attribut gibt an, wie das Dateisystem von der virtuellen Maschine verwendet wird. Für unsere Zwecke, reicht der Standardwert `type="cow"`.

<**mem**> Gibt an, wieviel RAM man der virtuellen Maschine zusichern möchte. Über Suffixe wie 'k', 'K', 'm' und 'M' gibt man an, ob es sich um Kilobytes oder Megabytes handelt. Der Standardwert ist auf 32M, also 32 Megabyte gesetzt.

<**kernel**> Da jeder virtuelle Rechner hochfahren muss, braucht er auch einen Kernel den er booten soll. Gibt man hier keinen Pfad zu einer Kernel-Datei an, so wird der Kernel aus `<default_kernel>` verwendet. Einschränkung für den manuell gesetzten Pfad sind, dass der Name des kernels auf jeden Fall das Wort `linux` enthalten muss.

<**boot**> Wie auf realen Rechnern auch, kann man einem Kernel Bootparameter mitgeben. Diese Bootparameter werden hier eingetragen, müssen aber valid sein, da sie vom vnumlparser nicht kontrolliert werden. Innerhalb von `<boot>` kann der `<con0>`- und der `<xterm>`-Tag verwendet werden. Mit ihm ist es möglich die Ausgaben der Maschine auf eine bestimmte Konsole umzuleiten. Eine X-Konsole kann auch angegeben, und über `<xterm>` genauer spezifiziert werden.

<**mng_if**> Dieses (optionale) Tag ermöglicht das Erzeugen von Management-Interfaces. Was Management-Interfaces sind, und wozu sie gebraucht werden, wurde bereits im ersten Seminarvortrag erklärt. Möchte man keine Management-Interfaces nutzen, so schreibt man `no` zwischen den öffnenden und schließenden Tag.

<**if**> Da die virtuellen Maschinen über ein Netzwerk miteinander kommunizieren, müssen erstmal Netzwerkschnittstellen definiert werden. Eine Netzwerkschnittstelle hat immer eine ID, die über das Attribut `id` vergeben wird. Die ID muss größer 0 sein, da die 0 für das Managementinterface reserviert ist. Für jede ID eines Netzwerkinterfaces wird auf der virtuellen Maschine ein Interface mit dem Namen `ethx` angelegt,

wobei x für die ID steht. Auch für den Host sind die Interfaces sichtbar, sie tauchen dort unter dem Namen `name-ethx` auf, wobei `name` für den Namen der virtuellen Maschine steht. An der Stelle werden jetzt die weiter oben definierten Netze in die Konfiguration mit einbezogen. Denn ein Interface muss immer an ein Netz gekoppelt werden. Natürlich muss der Name des Netzes das über das Attribut `net` gekoppelt wird, auch existieren. Es gibt noch 3 Untertags, die es erlauben das Interface mit MAC-Adresse und IP-Adresse zu konfigurieren:

<**mac**> Dieses Tag überschreibt die Zuweisung durch den globalen Tag `<automac>`, wenn er gesetzt wird. Um Problemen aus dem Wege zu gehen, sollten `fe:fd:` als die ersten beiden Bytes der Adresse gesetzt werden. Außerdem sollte die MAC-Adresse einzigartig in allen gestarteten Szenarios sein.

<**ipv4**> Definiert eine IP-Adresse für das Interface. Über das Attribute `mask` kann auch noch eine Subnetzmaske angegeben werden, die standardmäßig eine Klasse C Subnetzmaske ist.

<**ipv6**> Analog zu `<ipv4>` kann hier eine IP-Adresse festgelegt werden, die aber im IPv6-Format sein muss. Die Subnetzmaske wird nicht über ein Attribut vergeben, sondern durch das Anhängen der Anzahl der Maskenbits in folgender Form: `3ffe:ffff::3/64` für eine 64 Bit Maske.

<**route**> Definiert eine statische Route, die beim Starten des Szenarios in der Routingtabelle der virtuellen Maschine gesetzt wird. Es ist nicht unbedingt notwendig die Route hier zu setzen, da der Befehl `route` auf allen virtuellen Maschinen installiert ist. Über diesen kann man, vorausgesetzt man hat sich über SSH eingeloggt, die Route dann auch nachträglich setzen. Die Routen die über dieses Tag gesetzt werden, sind immer Gatewayrouten. Man kann über das Attribut `type` zusätzlich angeben, ob es sich um IPv4-Routen (`inet`) oder IPv6-Routen (`inet6`) handelt.

<**forwarding**> Über diesen Tag wird das Weiterreichen von IP-Paketen anhand der Routing-Tabelle des virtuellen Rechners aktiviert.

<**filetree**> Über diesen Tag wird ein Host-Verzeichnis angegeben (relativ zum in `<basedir>` angegebenen Verzeichnis), das komplett in das Dateisystem des Zielrechners, also der virtuellen Maschine, kopiert wird. Wurde kein `<basedir>`-Tag angegeben, so handelt es sich hier um einen absoluten Pfad. Anwendungsgebiete sind z.B., wenn man eine komplett

vorbereitete Konfiguration aus `/etc` auf die virtuellen Rechner kopieren möchte. Es sind zwei Attribute erlaubt:

- `root`: Wohin soll der Verzeichnisbaum kopiert werden (auf dem virtuellen Rechner)
- `when`: Gibt an, ob das Verzeichnis beim Starten, beim Beenden oder bei beiden Aktionen in das Zielverzeichnis kopiert werden soll. Hier ein kleines Beispiel, dass auch die Verwendung von `<basedir>` verdeutlicht:

```
<basedir>/home/usr/vnuml</basedir>
...
<vm name="uml1">
   ...
   <filetree root="/etc">uml1</filetree>
   ...
</vm>
<vm name="uml2">
   ...
   <filetree root="/etc">uml2</filetree>
   ...
</vm>
```

  "uml1" und "uml2" sind also Unterverzeichnisse von `/home/usr/vnuml` und vermindern beim Verschieben von Verzeichnissen die Angabe der Ordnerpfade.

<exec> Seit VNUML 1.5 existiert das `<exec>`-Tag statt der `<start>` und `<stop>`-Tags. Während ein Szenario läuft ist es möglich über den `vnumlparser` und den -x Parameter Kommandosequenzen auszuführen. Dies geschieht durch Angabe des Sequenznamens und der XML-Datei die die Sequenz enthält. Also beispielsweise:
`vnumlparser.pl -x testseq@testdatei.xml`
Diese Kommandosequenzen werden auch in der VNUML-Datei definiert. Da es möglich ist mehrere Kommandosequenzen anzugeben, werden diese über das Attribut `seq` voneinander unterschieden. `seq` bekommt den Namen der Sequenz der frei wählbar ist. Zwei Ausnahmen sind die Sequenznamen start und stop. Diese ersetzen nämlich die eben erwähnten veralteten Tags und dienen dazu Kommandosequenzen automatisch beim Starten und beim Beenden auszuführen. Neben dem `seq`-Attribut gibt es noch das `type`-Attribut. Dieses bestimmt von welcher Art die Kommandosequenz ist. Setzt man hier den Wert auf `verbatim`

so wird “wortwörtlich“ das Kommando ausgeführt, wie es innerhalb der `<exec>`-Tags steht. Ist der Wert `file`, so gibt man einen Pfad zu einem Shellskript an (auf dem Host-System), welches dann ausgeführt wird. Ein Beispiel:

```
<exec seq="test" type="verbatim">/usr/bin/ping uml1</exec>
<exec seq="test" type="verbatim">killall ping</exec>
```

Ruft man nun die Sequenz mit dem Namen “test“ auf, so werden beide Befehle der Reihe nach ausgeführt, da sie beide denselben Namen haben. Man hätte dies auch mit

```
<exec seq="test" type="file">/usr/bin/meinskript.sh</exec>
```

erreichen können, indem man die beiden Befehle im Skript plaziert hätte.

Ein Beispiel der Konfiguration einer einfachen VM wäre:

```
<vm name="uml4">
   <filesystem type="cow">/usr/local/share/
           vnuml/filesystems/root_fs_tutorial
   </filesystem>

   <if id="1" net="Net1">
      <ipv4>10.0.1.2</ipv4>
   </if>

   <route type="inet" gw="10.0.1.1">
      default
   </route>

   <forwarding type="ip" />
</vm>
```

4.4 Hostkonfiguration mit <host>

In diesem structural-Tag geht es um die Hostkonfiguration. Dieser Tag ist optional und sollte nicht verwendet werden, wenn der Host nicht Teil der Simulation ist. Der `<host>`-Tag ist sehr ähnlich zum `<vm>`-Tag, es können die Tags `<route>, <forwarding>` und `<exec>` auf dieselbe Weise verwendet werden. Zusätzlich gibt es zwei Tags die nur hier verwendet werden dürfen:

- `<hostif>`: Diesen Tag kann man sich als Pendant zum `<if>`-Tag aus der Konfiguration der virtuellen Maschinen vorstellen. Die Unterschiede sind:
  - Das `net`-Attribut wird mit derselben Bedeutung verwendet, allerdings nimmt der Name des Interface den Namen des hier angegebenen Netzes an. Nimmt man z.B. `net="Netz0"`, dann heißt das Interface `Netz0`, so wie es `ifconfig` zeigen würde. Das `id`-Attribut wird demnach also nicht verwendet.
  - `<mac>` kann hier nicht verwendet werden (das Interface hat ja schon eine "hardwaremäßig" festgelegte MAC-Adresse).
  - `<ipv4>` und `<ipv6>` unterscheiden sich nicht.
- `<physicalif>`: Da man mit `<hostif>` die Konfiguration einer physikalischen Schnittstelle überschreibt und der vnumlparser diese beim Beenden des Szenarios nicht alleine wiederherstellt, muss eine Konfiguration innerhalb des `<physicalif>`-Tags gespeichert werden. Die Konfiguration erfolgt anhand der Attribute `type` (ipv4 oder ipv6), `name` (z.B. eth0), `ip`, `mask` und `gw` (Standardgateway).

5 Version 1.6

Schaut man sich die Tags der Version 1.6 an, so scheinen sie vordergründig nur einige Möglichkeiten mehr zur Konfiguration eines Szenarios zu eröffnen. Hintergründig allerdings, ermöglichen sie es Nutzern mit verschiedenen Nutzerrechten Szenarios zu starten, auch wenn dies mit Einschränkungen verbunden ist. Alle Versionen vor Version 1.6 waren sehr Root fokussiert. Man kann sagen, dass es gar nicht, oder nur mit Umwegen möglich war, ein Szenario ohne Root-Reche zu starten. Seit Version 1.6 gibt es nun 3 Möglichkeiten VNUML zu verwenden, abhängig davon, welche Benutzerrechte man hat:

1. **Beschränkte Userrechte** Damit kann man den vnumlparser ohne Rootrechte starten. Die größte Einschränkung ist, dass kein direkter Zugriff vom Host auf die virtuellen Maschinen möglich ist. Somit ist es auch nicht möglich, dass die virtuellen Maschinen z.B. Zugriff aufs Internet erhalten, indem man die Interfaces über ein `external`-Netz mit einer Schnittstelle des Hosts verbindet. Stattdessen öffnet sich für jede virtuelle Maschine ein Terminal (in aller Regel ist das xterm) auf dem Host, über den man sich dann direkt einloggen kann. Eine weitere Konsequenz dieser Zugriffseinschränkung ist, dass der vnumlparser

nicht mehr dafür verwendet werden kann Kommandosequenzen auf den Maschinen auszuführen.

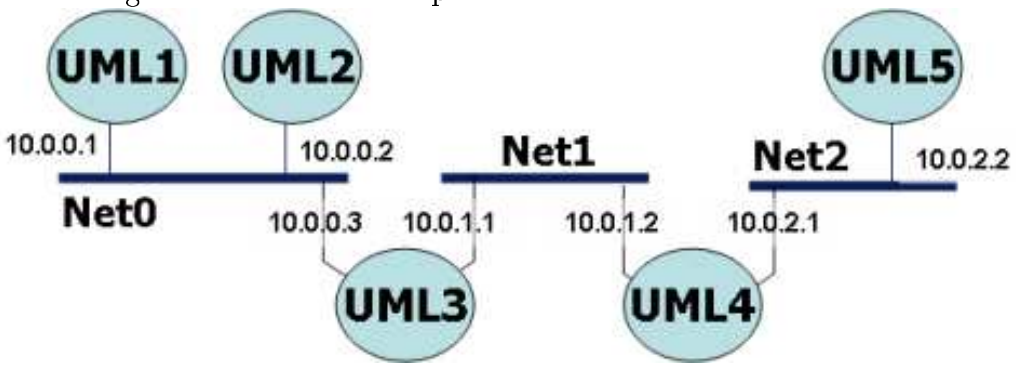
2. **Userrechte** Auch hier kann man den vnumlparser ohne Rootrechte starten. Was man bei *dieser* Möglichkeit VNUML zu benutzen aber aufbauen möchte, ist ein Management-Netz, das wir standardmäßig aus Version 1.5 gewohnt sind. Daraus folgert, dass wir jetzt auch Zugriff aufs Internet für die virtuellen Maschinen bereitstellen, indem wir den Host als Router/NAT verwenden (Layer 3 Verbindung). Auch Kommandosequenzen können jetzt über das Management-Netzwerk ausgeführt werden.

3. **Rootrechte** Hier wird der vnumlparser als Root gestartet. Nur mit Rootrechten ist es möglich dem vnumlparser zu erlauben den Host zu konfigurieren, ein sogenanntes privates Management-Netzwerk aufzubauen, Mapping der Namen der virtuellen Maschinen in der Datei `/etc/hosts`, und eine direkte Verbindung der virtuellen Maschinen an externe Netzwerke (Internet) mit denen der Host verbunden ist (Layer 2 Verbindung, realisiert über virtual bridges).

Natürlich brauchen wir für die verschiedenen Nutzungsmodi verschiedene Szenariokonfigurationen und verschiedene Arten den vnumlparser vorzubereiten und zu starten. Anhand der folgenden Beispiele lernen wir die in Version 1.6 neuen Tags kennen, und sehen sofort wofür sie gebraucht und wie sie eingesetzt werden.

5.1 Beschränke Userrechte

Abbildung 1: Aufbau eines Beispielszenarios mit beschränkten Userrechten



Und der dazugehörige gekürzte XML-Code:

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

<vnuml>
  <global>
    <version>1.6</version>
    <simulation_name>tutorial-lu</simulation_name>
    <automac/>
    <vm_mgmt type="none" />
    <default_filesystem type="cow">
          /usr/local/share/vnuml/filesystems/root_fs_tutorial
    </default_filesystem>
    <default_kernel>
          /usr/local/share/vnuml/kernels/linux
    </default_kernel>
  </global>
  <net name="Net0" mode="uml_switch" />
  <net name="Net1" mode="uml_switch" />
  <net name="Net2" mode="uml_switch" />
  <vm name="uml1">
    <boot>
      <con0>xterm</con0>
    </boot>
    <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">
    <boot>
      <con0>xterm</con0>
    </boot>
    <if id="1" net="Net0">
      <ipv4>10.0.0.2</ipv4>
    </if>
    <route type="inet" gw="10.0.0.3">default</route>
  </vm>
</vnuml>
```

Zunächst einmal fällt auf, dass es unter `<global>` ein neues Tag mit dem Namen `<vm_mgmt>` gibt. Über dieses Tag lässt sich die Art und das Management-

Netzwerk an sich konfigurieren. Über die verschiedenen Benutzerrechte hinweg, werden wir alle Attribute und Untertags von `<vm_mgmt>` kennenlernen. Über den hier angegebenen Typ `type=none` definieren wir, dass wir kein Management-Netzwerk wollen, da wir uns ja mit eingeschränkten Userrechten begnügen müssen.

Der nächste Punkt der auffällt, ist die Angabe von `mode` bei der Konfiguration der Netze. Bei Version 1.5 waren die Netze noch implizit im virtual bridge-Modus, virtuelle Brücken können aber nicht ohne Rootrechte kreiert werden. Stattdessen kümmert sich ein uml_switch-Prozess um die Verwaltung der Netze, dieser benötigt keine Rootrechte.

Bei den Tags der Version 1.5 haben wir bereits gesehen, dass man für jede virtuelle Maschine eine Shell auf dem Host öffnen lassen kann. Das war nicht dringend notwendig, da man sich ja auch über SSH einloggen konnte. Da wir hier kein Management-Netzwerk haben, brauchen wir diese Angabe einer Konsole (`<con0>`) dringend, um überhaupt Zugriff auf die Maschinen zu bekommen.

Was weiterhin fehlt ist die Konfiguration des Hosts. Ohne Rootrechte können wir weder die Interfaces des Hosts konfigurieren, noch die Namen der Maschinen auf ihre IP-Adressen mappen.

5.2 Benutzerrechte

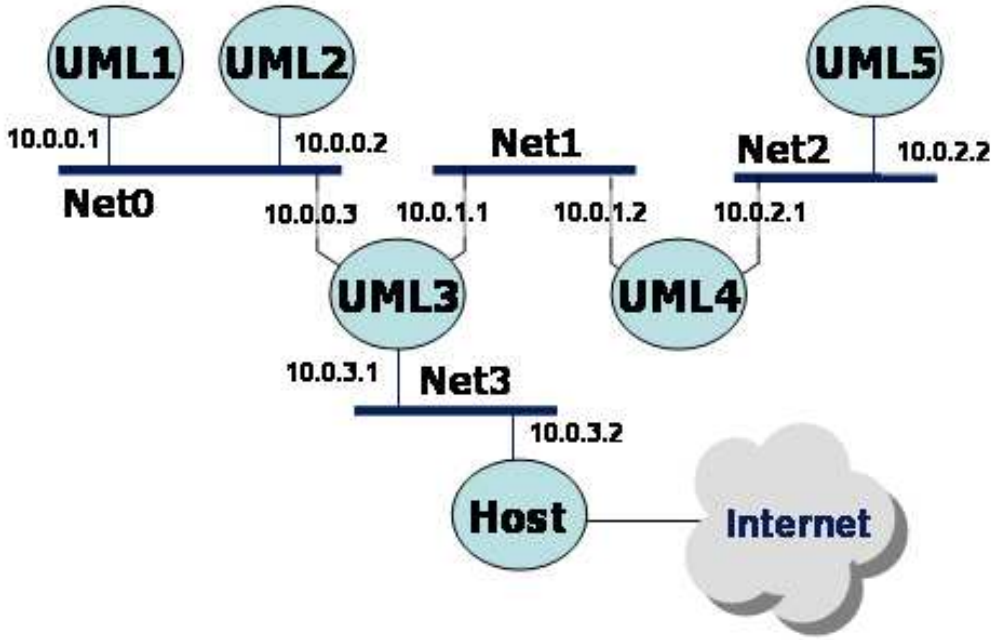
In diesem Modus kann man den vnumlparser zwar als konventioneller Benutzer starten, aber man braucht Rootzugriff (vielleicht über sudo) um einige Befehle als Superuser auszuführen. Dadurch wird es aber möglich die Simulation mit dem Host zu verbinden und Kommandosequenzen auf den virtuellen Rechnern auszuführen.

Wie man in Abb. 2 sehen kann, ist das Szenario eine kleine Variation des eben vorgestellten. Wir haben immer noch kein Management-Netzwerk, aber eine Verbindung zum Host. Der dazugehörige gekürzte XML-Code, der die Unterschiede zum vorherigen Szenario beinhaltet:

```
...
  <net name="Net0" mode="uml_switch" />
  <net name="Net1" mode="uml_switch" />
  <net name="Net2" mode="uml_switch" />
  <net name="Net3" mode="uml_switch"
            sock="/var/local/run/vnuml/Net3.ctl" />
...
  <vm name="uml3">
    <boot>
```

```
      <con0>xterm</con0>
    </boot>
    <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 type="inet" gw="10.0.1.2">10.0.2.0/24</route>
    <forwarding type="ip" />
  </vm>
...
```

Abbildung 2: Aufbau eines Beispielszenarios mit beschränkten Userrechten

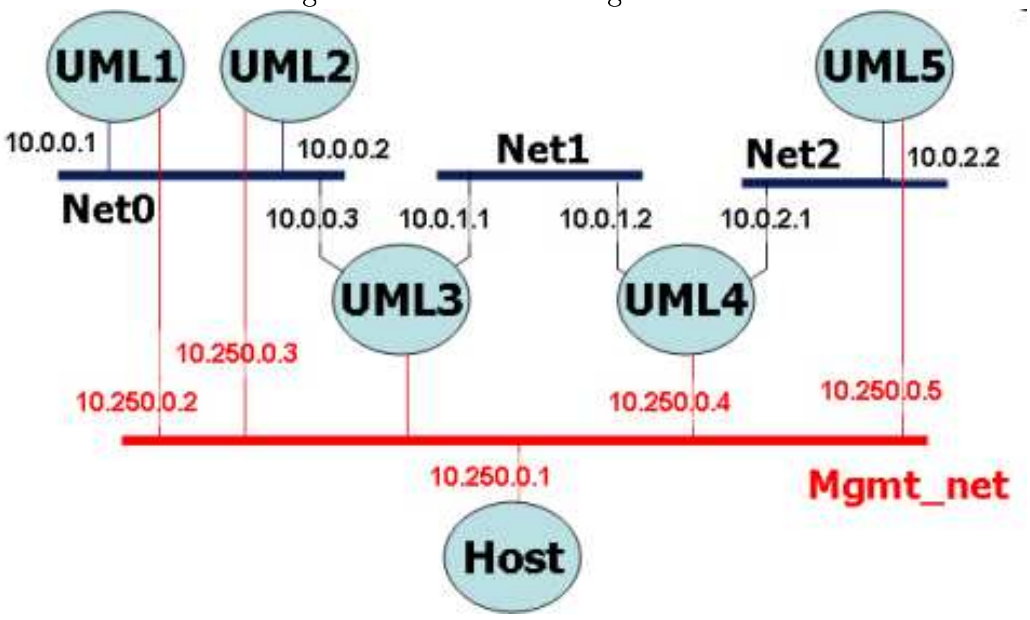


Die wichtigen Zeilen sind die der Konfiguration von "Net3" und die Verbindung der virtuellen Maschine uml3 an dieses Netz über das Interface mit der id=3. Das Netz "Net3" hat hier nämlich ein `socket`-Attribut, dass einen Pfad zu einem generierten Socket auf dem Host besitzt. Zuvor wur-

de mit Rootrechten ein TAP-Device auf dem Host erstellt, und mit einer IP-Adresse samt Netzmaske und Standardroute versehen. Im Anschluss wird der uml_switch-Prozess (auch als Root) gestartet, und bindet das TAP-Device an einen Socket der daraufhin kreiert wird. Der Benutzer, der ja nur Standardbenutzerrechte hat, muss Lese- und Schreibzugriff auf diesen Socket haben, wenn er den vnumlparser startet um das Szenario hochzufahren. Über diesen Socket kommuniziert nun die virtuelle Maschine mit dem Host. Welche Befehle genau für diese Konfiguration nötig sind, entnimmt man an dieser Stelle bitte der VNUML 1.6 Referenz.

Wie Eingangs des Kaptiels erwähnt, soll es möglich sein mit eingeschränkten Benutzerrechten ein Management-Netzwerk aufzubauen:

Abbildung 3: Aufbau mit Management-Netzwerk



Die wichtigen XML-Codestellen zeigen die Konfiguration des Management-Netzes:

```
<vm_mgmt type="net" network="10.250.0.0" mask="24">
     <mgmt_net sock="/var/local/run/vnuml/Mgmt_net.ctl"
                  hostip="10.250.0.1"/>
</vm_mgmt>
```

Als `type` des Management-Netzes verwenden wir diesmal "net". Ein weiterer Typ "private", erstellt eine peer-to-peer-Verbindung zwischen dem Host und jeweils einer virtuellen Maschine (das setzt aber voraus, dass der vnumlparser als Root gestartet wurde). Die Alternative dazu ist, dass man über `type=net`

ein uml_switch-Netzwerk verwendet, dann aber auch das `network` und `mask`-Attribut setzt, damit der vnumlparser einen Netzwerkadressenbereich für die Vergabe der Adressen an die virtuellen Maschinen verwenden kann. Nur wenn man den Typ "net" verwendet ist es erlaubt, den Untertag `<mgmt_net>` zu verwenden. Dieser wird gebraucht um das uml_switched-Netzwerk weiter zu konfigurieren. Da wir auch für das Management-Netzwerk eine Verbindung zum Host brauchen, muss wie schon im ersten Beispiel des "Benutzerrechtemodus" ein TAP-Device erstellt werden, welches dann über den Socket mit den virtuellen Maschinen kommunizieren kann. Die IP wird über das Attribut `hostip` festgelegt werden, und muss mit der vergebenen IP auf dem Host übereinstimmen. Auch hier verweise ich für weitere Konfigurationsdetails auf die Referenz.

Die Kommandosequenzen die ja nun auch verschickt werden können, werden intern von VNUML über SSH verschickt.

5.3 Rootrechte

Um nochmal den Unterschied zwischen den Benutzerrechten und vollem Root-Zugriff zu verdeutlichen, wollen wir uns die XML-Datei ansehen, die die Abbildung 2 mit Rootrechten modelliert:

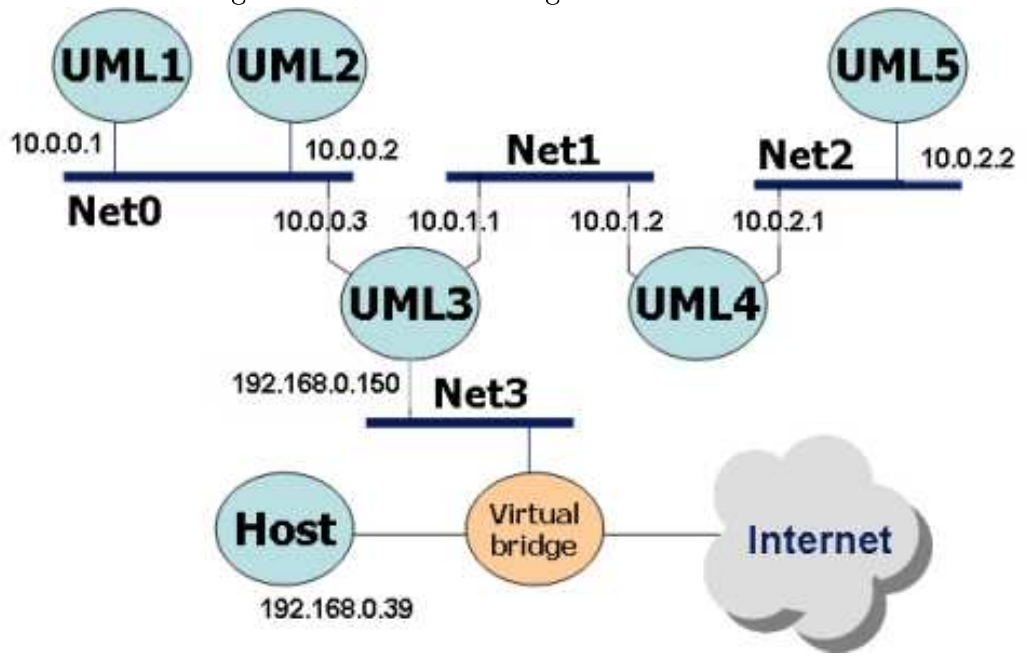
```
...
<net name="Net0" mode="uml_switch" />
<net name="Net1" mode="uml_switch" />
<net name="Net2" mode="uml_switch" />
<net name="Net3" mode="uml_switch" />
...
<host>
    <hostif net="Net3">
       <ipv4>10.0.3.2</ipv4>
    </hostif>
    <route type="inet" gw="10.0.3.1">10.0.0.0/16</route>
</host>
...
```

Wir haben jetzt ganz normale Netze, ohne eine Socketangabe, können jetzt aber auch den Host konfigurieren, und ihn wie eine virtuelle Maschine an "Net3" binden. Um die Erstellung eines TAP-Device, eines Sockets und das Starten des uml_switch-Prozesses kümmert sich der vnumlparser Dank der Rootrechte im Hintergrund.

Ein weiteres Beispiel welche neuen Möglichkeiten man mit Rootrechten hat, ist eine Layer 2 Verbindung einer virtuellen Maschine zu einem

externen Netzwerk des Hosts. Dies wird realisiert über die Attributangabe `type="virtual_bridge"` bei der Definition der Netze. Zusätzlich wird der Name der physischen Netzwerkschnittstelle des Hosts im `external`-Attribut angegeben. Das Beispiel verbindet den Rechner uml3 direkt mit dem externen Netzwerk des Hosts, und setzt für ihn die IP 192.168.0.150. Natürlich muss diese IP in einem realen Szenario auf einen Wert gesetzt werden, der im externen Netzwerk Sinn macht.

Abbildung 4: Direkte Verbindung über eine virtuelle Brücke




```
...
<net name="Net0" mode="uml_switch" />
<net name="Net1" mode="uml_switch" />
<net name="Net2" mode="uml_switch" />
<net name="Net3" mode="virtual_bridge" external="eth0"/>
...
<vm name="uml3">
    <boot>
      <con0>xterm</con0>
    </boot>
    <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>192.168.0.150</ipv4>
    </if>
    <route type="inet" gw="10.0.1.2">10.0.2.0/24</route>
    <forwarding type="ip" />
  </vm>
...
<host>
    <hostif net="Net3">
       <ipv4>192.168.0.39</ipv4>
    </hostif>
    <physicalif name="eth0" ip="192.168.0.39"
           mask="255.255.255.0" gw="192.168.0.1" />
    <route type="inet" gw="192.168.0.1">default</route>
  </host>
...
```

5.4 Weitere neue Tags in VNUML 1.6

Das neue `<vm_mgmt>`-Tag haben wir ja bereits kennengelernt. Ein Untertag von `<vm_mgmt>`, der zwar kurz angesprochen aber nicht gezeigt wurde, ist `<host_mapping>`. Die Namen der virtuellen Maschinen werden hierbei auf ihre IP-Adressen gemappt. Da dafür die Datei `/etc/hosts` editiert werden muss, sind Rootrechte nötig um diesen Tag in der Konfiguration verwenden zu dürfen.

Als neue Funktionalität von VNUML 1.6 ist das Erstellen von Benutzern samt Zugehörigkeit auf virtuellen Maschinen hinzugekommen. Über das Attribut `username` wird ein Benutzername, und über `group` eine "effektive" Gruppe festgelegt. Es ist möglich weitere Gruppenzugehörigkeiten über das Untertag `<group>` anzugeben, die effektive Gruppe muss Teil eines der `<group>`-Tags sein. Gibt man keine Gruppe an, so wird der Systemstandard auf der virtuellen Maschine verwendet. Auch `<ssh_key>` ist ein Untertag, der einen SSH-Schlüssel in die Benutzerumgebung des definierten Benutzers einfügt.

Fasst man diese Änderungen mit ein paar kleinen weiteren zusammen, kommt man zu folgender Übersicht:

- Es gibt ein neues, wichtiges und mächtiges Tag `<vm_mgmt>`

- Man kann Benutzer auf den virtuellen Maschinen über `<user>` erstellen
- Der Tag `<ip_offset>` aus Version 1.5 wird durch die Attribute `network`, `mask` und `offset` im `<vm_mgmt>`-Tag ersetzt
- Das `version`-Attribut aus `<ssh_key>` wurde in den Tag `<ssh_version>` ausgelagert
- Version 1.5 impliziert noch `type="private"` als Management-Netzwerk, jetzt frei wählbar
- `<host_mapping>` ist von `<global>` nach `<vm_mgmt>` gewandert

6 Fazit

VNUML bietet umfangreiche Konfigurationsmöglichkeiten für virtuelle Maschinen, und die virtuellen Netze mit denen sie sich verbinden lassen. Kontinuierlich werden Tag-Strukturen verbessert, verändert, und um neue Funktionalitäten erweitert. Die Entwickler richten sich nach Anwenderwünschen, der Benutzer- und Benutzerrechtephilosophie von Linux, und ermöglichen es Szenarien mit unterschiedlichen Rechten und in unterschiedlichen Modi zu starten. Das erlaubt auch den Einsatz von VNUML in Umgebungen wo Benutzer eben nicht Vollzugriff auf den Rechner haben dürfen, was beispielsweise an einer Universität der Fall ist. Ob zum "Spielen", ein wenig Testen, oder für wissenschaftliche Projekte: VNUML bietet eine solide und umfangreiche Plattform auf dem Gebiet von Netzwerkstrukturen, der damit verbundenen Protokolle und Anwendungen.

Literatur

[VNUML 1.5 Tutorial] *http://jungla.dit.upm.es/ vnuml/doc/1.5/tutorial*

[VNUML 1.5 User manual] *http://jungla.dit.upm.es/ vnuml/doc/1.5/user*

[VNUML 1.5 User reference] *http://jungla.dit.upm.es/ vnuml/doc/1.5/reference*

[VNUML 1.6 Tutorial] *http://jungla.dit.upm.es/ vnuml/doc/1.6/tutorial*

[VNUML 1.6 User manual] *http://jungla.dit.upm.es/ vnuml/doc/1.6/user*

[VNUML 1.6 User reference] *http://jungla.dit.upm.es/ vnuml/doc/1.6/reference*

[Arndt, Richard (2005)] *VNUML-Virtual Network User Mode Linux. Seminar Routing WS04/05*