INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Ein Rechtemodell für VIFS

Gerhard Gröschl

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Ein Rechtemodell für VIFS

Gerhard Gröschl

Aufgabensteller:	Dr. Vitalian A. Danciu
Betreuer:	Dr. Vitalian A. Danciu Tobias Guggemos
Abgabetermin:	20. Februar 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Februar 2018

.....
(Unterschrift des Kandidaten)

Abstract

Aufgrund der Ausmaße rechnerbasierter Infrastrukturen und deren Verwaltungsaufwand wird nach Möglichkeiten gesucht, zentrale Verwaltungs- sowie Steuerungsmöglichkeiten für diese Strukturen zu erzeugen. Ein Ansatz, der eine elegante Lösung für dieses Problem bereitstellt, existiert bereits unter dem Namen VIFS - Virtual Infrastructure Filesystem [Dan16b]. VIFS bietet die Möglichkeit zur Abbildung physischer sowie virtueller Infrastrukturen in ein Dateisystem. Durch VIFS können komplexe Konstrukte in hierarchische Verzeichnisse und zugehörige Operationen in Dateien abgebildet werden. Ein Baustein der diesem System jedoch noch fehlt, ist ein entsprechendes Rechtemodell. Dieses Rechtemodell muss in der Lage sein, Zugriffe auf die Abbildungen domänenbasierter Strukturen zu kontrollieren und eine flexible Verwaltung der Zugriffsrechte zu ermöglichen. Diese Arbeit widmet sich dem Entwurf eines Rechtemodells sowie der Erzeugung eines geeigneten Prototyps.

Inhaltsverzeichnis

1 Einleitung

Das “Virtual Infrastructure Filesystem”, (kurz VIFS), ist ein Managementsystem, welches physische sowie virtuelle Infrastrukturen in ein Dateisystem abbildet und Managementoperationen zur Verfügung stellt. Dabei werden Komponenten auf Verzeichnisse und Operationen auf Dateien abgebildet. Ziel dieses Dateisystems ist es, durch die zu verwaltende Struktur mit beliebigen Dateisystemmanagern und Terminals navigieren und sie durch einfache Aktionen (Ändern des Modifikationszeitstempels) steuern und verwalten zu können. Dabei können beliebige Infrastrukturen - uneingeschränkt ihrer Heterogenität, Breite oder Tiefe - abgebildet und verwaltet werden. Viele Operationen sind bereits in VIFS integriert und reduzieren dadurch den Verwaltungsaufwand einer Infrastruktur erheblich. Bisher gilt dies jedoch lediglich für einen einzigen Benutzer, da VIFS zum aktuellen Zeitpunkt noch keine geeignete Rechteverwaltung zur Verfügung stellt.

Das MNM-Team (Munich Network Management) der Ludwig-Maximilian-Universität (LMU) führt jährlich mehrere Praktika mittels geschachtelten virtuellen Umgebungen durch. Ein Beispiel für ein solches Praktikum ist das in Kapitel 3 als Szenario verwendete Rechnernetzepraktikum. Ein weiteres Beispiel, welches sich ebenfalls geschachtelte virtuelle Umgebungen zu Nutze macht, ist das IT-Sicherheitspraktikum. Die Verwaltung dieser Umgebungen erzeugt einen nicht zu vernachlässigenden Management-Aufwand, welcher mittels VIFS erheblich reduziert werden könnte. Als Anwendungsbeispiel dieser Arbeit soll das Rechnernetzepraktikum der LMU dienen. Bei diesem Praktikum lernen Studenten Rechnernetze zu konfigurieren und zu verwalten. Zeitgleich muss die virtuelle Infrastruktur von Administratoren gewartet und konfiguriert werden. VIFS kann bereits zur teilweisen Verwaltung dieses Praktikums eingesetzt werden und würde durch eine Mehrbenutzer-Unterstützung wesentlich dazu beitragen, den Verwaltungsaufwand zu verringern. Weil jedoch noch kein geeignetes Rechtemodell für VIFS existiert, kann es nur von einer Person eingesetzt werden. Da größere Infrastrukturen selten von einer einzelnen Person verwaltet und benutzt werden, wird ein Berechtigungsmodell benötigt, das sicherstellt, welcher Benutzer an welchen abgebildeten Systemen welche Operationen ausführen darf. Aus diesem Grund zielt diese Arbeit darauf ab, ein Rechtemodell zu finden, das VIFS die Steuerung und Verwaltung von Infrastrukturen ermöglicht, welche durch mehrere Domänen in separate Verantwortungsbereiche gegliedert sind und den Gliederungen entsprechend eine Zugriffskontrolle zur Verfügung stellt. Hierbei ist es von besonderem Interesse, Zuständigkeitsbereiche strikt zu trennen und dabei dennoch wichtige Managementoperationen wie beispielsweise domänenübergreifende Delegation zu ermöglichen. Aus diesem Grund wurde eine Schnittstelle für VIFS definiert, welche alle im RNP benötigten Operationen abdeckt, um darauf basierend ein Rechtemodell zu erzeugen.

Um das Ziel dieser Arbeit zu erreichen, wurden zuerst in Kapitel 2 die benötigten Grundlagen bezüglich Virtualisierung und der Funktionsweise von VIFS untersucht. Desweiteren wurde ein genauer Blick auf die wichtigsten Merkmale der POSIX-Schnittstelle, sowie den Aufbau des OSI-Managementkonzepts geworfen. Um eine fundierte Grundlage für das benötigte

Domänenkonzept und die daraus resultierenden Anforderungen an das zu erstellende Rechtemodell zu erhalten, wurde in Kapitel 3 ein Szenario des bereits angesprochenen Rechnernetzepraktikums erstellt und analysiert. Kapitel 4 widmet sich den resultierenden Anforderungen an das Rechtemodell. Diese wurden anhand der entstandenen Schnittstelle, welche zur Verwaltung von beliebigen Infrastrukturen benötigt wird, sowie dem geschilderten Szenario eruiert. Kapitel 5 beschäftigt sich mit zwei möglichen Lösungsansätzen. Der erste gilt der Erzeugung des theoretischen Modells und wird in Kapitel 5.1 beschrieben. Dieses Rechtemodell würde zwar allen eruierten Anforderungen am besten entsprechen, lässt sich aber aufgrund von Inkompatibilität seitens der verwendeten Software-Komponenten nicht realisieren. Durch die dabei gewonnenen Erkenntnisse wurde in Kapitel 5.2 ein den Anforderungen entsprechendes Modell mittels der traditionellen POSIX-Zugriffskontrolle abgebildet. Diese Abbildung wurde in Kapitel 6 umgesetzt. Der daraus resultierende Prototyp kann die Anforderungen erfüllen und bietet die benötigte Funktionalität, um den Betrieb eines mehrbenutzerfähigen VIFS zu ermöglichen. In Kapitel 7 befindet sich eine Zusammenfassung sowie ein Ausblick für zukünftige Arbeiten.

2 Grundlagen

Dieses Kapitel widmet sich den für diese Arbeit benötigten Grundlagen. Zu Beginn gibt es einen Überblick über Virtualisierung. Im darauf folgenden Unterkapitel wird eine Übersicht über die Arbeitsweise von VIFS gegeben. Kapitel 2.3 geht auf die zur Verfügung stehenden Möglichkeiten und Limitierungen des vorausgesetzten POSIX-Dateisystem-Standards ein, um mögliche Schwierigkeiten bei der Abbildung der Management-Domänen im Vorfeld zu erkennen. Als Abschluss dieses Kapitels wird das OSI-Management Modell vorgestellt, welches bei der Unterteilung der vorhandenen Domänen behilflich ist.

2.1 Virtualisierung

Durch die äußerst effiziente Nutzung von Hardware mittels Virtualisierung steigt die Nachfrage nach virtualisierten Infrastrukturen enorm. Serviceprovider müssen die Hardwarebasis für ihre Dienste nicht mehr selbst unterhalten und mieten stattdessen auf ihre Bedürfnisse maßgeschneiderte virtuelle Maschinen (VM) an. Diese Entwicklung fördert einen niedrigeren Rohstoffverbrauch und wirkt zusätzlich durch die flexible Handhabbarkeit der einzelnen VM dem Internet Ossification Problem entgegen [APST05]. Durch die zusätzliche Möglichkeit der Verschachtelung virtueller Maschinen entstehen gekapselte hierarchische Topologien [Dan16a], welche Unterteilungen in Management-Domänen vereinfachen. Um zu jedem Zeitpunkt die Ebene einer VM beschreiben zu können, wird die Nomenklatur aus [Dan16a] übernommen. Dabei stellt eine physische Instanz stets eine 0-VM dar. Wird von dieser 0-VM eine virtuelle Maschine gehostet, so spricht man von einer 1-VM. Sollte diese 1-VM wiederum ein Host für weitere VM sein, so befinden sich diese weiteren VM auf Level 2 und werden dahingehend als 2-VM beschrieben. Die Überschaubarkeit der Systeme kann durch Virtualisierung in Mitleidenschaft gezogen werden. Verschiedene Virtualisierungsanbieter wie beispielsweise XEN oder KVM [xen, kvm] nutzen unterschiedliche Technologien, um zweckmäßige Vorteile zu bieten. Kompatibilität ist hierbei nicht immer gewährleistet und so kann es vorkommen, dass mehrere Virtualisierungsanbieter auf derselben Plattform zum Einsatz kommen, was die Komplexität ebenfalls steigert. Das Management solcher hierarchischen Strukturen ist somit nicht trivial und eine zentrale Steuerungs- sowie Verwaltungsmöglichkeit aller relevanten Instanzen ist erstrebenswert.

2.2 VIFS als Managementplattform

VIFS bildet die Quellinfrastruktur als Dateisystem auf den Zielhost ab und stellt via “Push-Button”-Dateien diverse Aktionen zur Verfügung. Beispiele für solche Aktionen sind in Tabelle 2.1 aufgelistet. Um Verbindung zu den Quellinfrastrukturen aufzubauen, benötigt VIFS Zugang zu einem ausreichend berechtigten Benutzerkonto - im bisherigen Vorgehen zum “root”-Nutzer. Man kann also davon ausgehen, dass ohne Zugriffskontrolle auf dem gemounteten Dateisystem, jeder Nutzer der Zugriff darauf hat, alle angebotenen Operationen auf

Tabelle 2.1: Push-Button-Beispiel

Aktion	Push-Button
Neustarten eines Rechners	+REBOOT
Herunterfahren eines Rechners	+SHUTDOWN
Löschen einer VM	+DESTROY
Pausieren einer VM	+PAUSE
VM aus Ruheszustand holen	+RESUME
VM in Ruhezustand versetzen	+SUSPEND
VM aus Pausezustand holen	+UNPAUSE

allen verbundenen Systemen ausführen kann. Eine VIFS-Instanz soll jedoch von mehreren Benutzern verwendet werden können und so ist es notwendig, eine Zugriffskontrolle in VIFS zu integrieren, welche alle dem Quellsystem entsprechenden Management-Domänen voneinander abgrenzt.

VIFS wird bereits zur Verwaltung des in Kapitel 3 vorgestellten Rechnernetzepraktikums der LMU eingesetzt. Aufgrund der fehlenden mehrbenutzertauglichen Zugriffskontrolle wird es jedoch aktuell nur vom Betreuer des Praktikums verwendet. Ein Auszug der von VIFS anhand dieses Praktikums erzeugten Dateisystemstruktur ist in Abbildung 2.1 zu sehen. Das Präfix '_' identifiziert Elemente dieser Auflistung als Informationsdateien. Zusätzlich enthalten alle Ordner weitere versteckte Informationsdateien ('.'-Präfix), welche jedoch der Übersichtlichkeit halber nicht gezeigt werden. Mit dem Präfix '+' versehene Elemente stellen die zuvor erklärten Push-Buttons dar. Elemente ohne diese Präfixe sind Verzeichnisse. Wichtige Informationsdateien sind beispielsweise die _type-Dateien. Diese Dateien dienen zur Identifizierung des Typs, welcher durch den beinhaltenden Ordner repräsentiert wird. Die Datei _type im Verzeichnis router01 beinhaltet hierbei das Stichwort 'vm', während die _type-Datei im Verzeichnis vif11.0 das Stichwort 'vif' (virtual interface) beinhaltet. Dem Typ entsprechend werden verschiedene Operationen angeboten. So kann durch eine Änderung des Zeitstempels der Datei +REBOOT mittels "touch"-Befehl die durch den beinhaltenden Ordner repräsentierte Maschine neu gestartet werden. Bei einem Interface kann durch dieselbe Vorgehensweise die entsprechende Schnittstelle aktiviert (+UP) oder deaktiviert (+DOWN) werden. Zusätzlich zu den bereits vorhandenen Operationen könnten noch weitere Operationen implementiert werden, wie beispielsweise das Öffnen eines Terminals auf dem entsprechenden Rechner. Auch das Herstellen einer Netzverbindung mittels virtuellem Switch und entsprechender Anbindung zweier Maschinen unter Verwendung eines Symlinks ist bereits teilweise integriert. Eine Zuordnung von bestehenden VIFS-Operationen zu Operationen auf den betreffenden Systemen ist in Tabelle 2.2 zu sehen. Um den gesamten Umfang des RNP abzudecken und eine zweckmäßige Verwendung von VIFS für alle Praktikumsteilnehmer und Veranstalter zu ermöglichen, werden weitere Operationen und vor allem eine Zugriffskontrolle benötigt. Da das von VIFS mittels FUSE [fus] gemountete Dateisystem auf POSIX basiert, folgt eine Untersuchung der Möglichkeiten die POSIX zur Verfügung stellt.

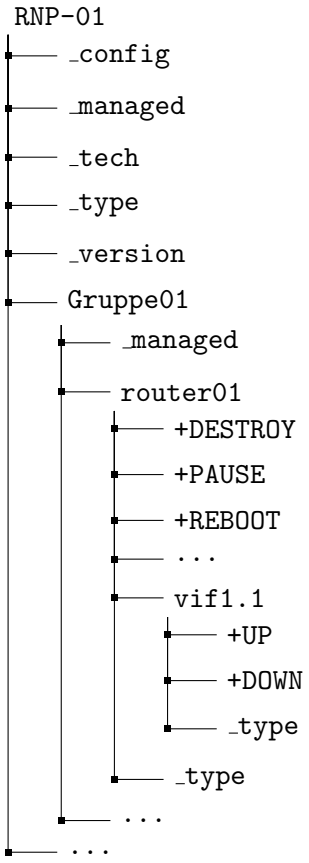

Abbildung 2.1: Auszug des VIFS-Dateisystems anhand des RNP

Tabelle 2.2: Auszug von VIFS-Operationen

Operation	VIFS-Operation
router01 herunterfahren	touch .../RNP-01/Gruppe01/router01/+SHUTDOWN
router01 neustarten	touch .../RNP-01/Gruppe01/router01/+REBOOT
pc01 eth1 ausschalten	touch .../RNP-01/Gruppe01/pc1/vif11.1/+DOWN
pc01 eth1 einschalten	touch .../RNP-01/Gruppe01/pc1/vif11.1/+UP
router04 pausieren	touch .../RNP-02/Gruppe04/router04/+PAUSE
router04 fortfahren	touch .../RNP-02/Gruppe04/router04/+UNPAUSE
pc03 in Ruhemodus versetzen	touch .../RNP-01/Gruppe01/router01/+SUSPEND
pc03 aus Ruhemodus fortfahren	touch .../RNP-01/Gruppe01/router01/+RESUME
router02 löschen	touch .../RNP-01/Gruppe01/router02/+DESTROY
router01 der Gruppe01 an Switch5 anbinden	ln -s .../RNP-01/Gruppe01/router01 router01 in Ordner .../RNP-01/Gruppe01/Switch5
Gruppe01 von RNP-01 nach RNP-02 migrieren	mv .../RNP-01/Gruppe01 .../RNP-02

2.3 POSIX

Da die Abbildung der virtuellen Infrastrukturen in ein Dateisystem sowie die Integration dieses Dateisystems in das Dateisystem von VIFS-Nutzern nahezu unabhängig vom verwendeten Betriebssystem funktionieren soll, ist die POSIX-Konformität des Rechtemodells eine Grundvoraussetzung dieser Arbeit. Um diese Voraussetzung zu erfüllen, findet an dieser Stelle eine Analyse der Vorgaben von POSIX statt. Eine umfangreichere Beschreibung der Elemente eines POSIX-Dateisystems kann unter [posa] nachgelesen werden. Das Wissen über die hierarchische Struktur eines POSIX-Dateisystems sei an dieser Stelle vorausgesetzt. Wird in diesem Unterkapitel von Zeichen gesprochen, sei erwähnt, dass POSIX eine abgewandelte 7-Bit ASCII-Kodierung verwendet [posb]. Wird in diesem Unterkapitel die Bezeichnung 'Element' gewählt, handelt es sich um eine Datei oder ein Dateiverzeichnis, wobei die Unterscheidung an dieser Stelle keine Rolle spielt. Dateien können weiters in verschiedene Kategorien mit verschiedenen Eigenschaften unterteilt werden, wobei für diese Arbeit lediglich Textdateien, Scripte, Binärprogramme und symbolische Links eine Rolle spielen. Letztere verfügen über keine eigenen Dateirechte, sondern repräsentieren lediglich den Pfad zu Elementen, welche ihre eigenen Dateirechte umsetzen.

Zunächst ist eine Auflistung der Limitierungen seitens POSIX sinnvoll:

- Die Pfadnamen von POSIX-Dateien können maximal 1023 Byte lang sein
- Dateinamen selbst können maximal 255 Zeichen lang sein
- Gruppen- sowie Besitzernamen unterliegen einer Begrenzung von 32 Zeichen
- Jedem Element kann nur ein Besitzer und eine Gruppe zugeteilt werden

Elementar für diese Arbeit ist die POSIX-Zugriffskontrolle, mit welcher die Realisierung der Zugriffskontrolle für mehrere Benutzer auf dem VIFS-Host erfolgen soll. Diese unterteilt die drei Benutzerklassen 'Besitzer', 'Gruppe' und 'Andere', welche jedem Dateisystemelement

zugeteilt werden. Des Weiteren ist jede dieser Benutzerklassen im Besitz einer Kombination aus den grundlegenden Rechten 'Lesen'(r), 'Schreiben'(w) und 'Ausführen'(x). Zusammen mit den erweiterten Rechten 'Setuid', 'Setgid' und dem sogenannten 'Sticky Bit' werden die Rechte den Benutzerklassen für jedes Dateisystemelement zugewiesen. Ein nicht vorhandenes Recht wird durch einen Bindestrich dargestellt.

Als nächstes werden die Auswirkungen von POSIX-Rechten auf Verzeichnisse etwas genauer untersucht. Hierbei werden einige Unterschiede zwischen Dateien und Dateiverzeichnissen ersichtlich. Um es überschaubar zu halten wird zuerst die Benutzerklasse 'Andere' betrachtet. Dabei kann man davon ausgehen, dass Besitzer und Gruppe 'root' zugeordnet sind. Deren Rechte-Attribute sind an dieser Stelle nicht relevant. Wird über Benutzer gesprochen, ist stets ein 'normaler' Benutzer, also ohne erhöhte Systemrechte, gemeint. Das aktuelle Verzeichnis erlaubt Lese- und Ausführ-Rechte. Welche Aktionen Benutzer unter diesen Umständen in/mit Verzeichnissen ausführen dürfen, wird in 2.3 aufgelistet.

zugeteilte Dateirechte	Befugnisse des Benutzers
- - -	
- - x	Verzeichnis betreten
- w -	
- w x	Verzeichnis betreten, in Verzeichnis schreiben, alle Dateien in Verzeichnis umbenennen und löschen (Dateiname muss bekannt sein), neue Dateien schreiben
r - -	Verzeichnisinhalt auslesen (ohne Anzeige zugehöriger Rechte)
r - x	Verzeichnis betreten, Verzeichnis auslesen, enthaltene Dateien abhängig von jeweiligen Rechte bearbeiten
r w -	Verzeichnis auslesen (ohne Anzeige zugehöriger Rechte, nicht betreten)
r w x	Verzeichnis betreten, auslesen, in Verzeichnis schreiben, enthaltene Dateien umbenennen und löschen, neue Dateien erzeugen, enthaltene Dateien abhängig von Dateirechten bearbeiten

Tabelle 2.3: Rechte und Befugnisse an Verzeichnissen

Wie man hier bereits sieht, erzeugt jedes Bit für sich allein überschaubare Rechte. Die vorhin gewählten "r-x"-Rechte für Verzeichnisse kristallisieren sich als sehr nützlich heraus. Sie delegieren die Verantwortung für die Verwendung enthaltener Dateien und Verzeichnisse an diese weiter, schützen sie jedoch zusätzlich vor dem Umbenennen und Löschen. Dateien verhalten sich nur teilweise ähnlich. Tabelle 2.4 zeigt die Befugnisse an Dateien. Es wird schnell klar,

zugeteilte Dateirechte	Befugnisse des Benutzers
- - -	
- - x	ausführen falls Binärprogramm
- w -	in Datei schreiben, [ohne zu lesen (> & >>)]
- w x	in Datei schreiben, [ohne zu lesen (> & >>)], ausführen falls Binärprogramm
r - -	Dateiinhalt lesen, indirekt ausführen
r - x	Dateiinhalt lesen und ausführen
r w -	Dateiinhalt lesen, in Datei schreiben, indirekt ausführen
r w x	Dateiinhalt lesen, ausführen, in Datei schreiben

Tabelle 2.4: Rechte und Befugnisse an Dateien

dass Rechte bei Verzeichnissen und Dateien unterschiedliche Auswirkungen haben. Ein Ver-

zeichnis kann mittels '- - x' betreten werden, ein Binärprogramm kann eigenständig vom Benutzer ausgeführt werden, an einem Script oder anderen Dateien kann keine Operation initialisiert werden. Mittels '- w -' kann, ohne den Inhalt lesen zu können, in eine Datei geschrieben werden (anhängen oder überschreiben), an einem Verzeichnis mit denselben Rechten können keine Operationen ausgeführt werden. Zusätzlich ist noch auf Folgendes zu achten: Der Eigentümer des Elements darf, unabhängig von den Rechten des Elements, die Gruppe des Elements ändern, sofern er Teilnehmer der Zielgruppe ist und das beinhaltende Verzeichnis das x-bit für den Benutzer geltend macht. Grundlegend darf er, unabhängig von der Gruppe oder den Rechten, alle Rechte des Elements ändern. Einem Verzeichnis oder einer Datei kann jeweils nur ein Besitzer und eine Gruppe zugeordnet werden. Vorausschauend kann erwähnt werden, dass Gruppenkonstrukte unter POSIX nur so lange als Rollenträger funktionieren, solange sich die Zugriffsbereiche verschiedener Rollen nicht überschneiden. Einer Gruppe können beliebig viele Benutzer hinzugefügt werden, allerdings keine anderen Gruppen. Weiters bietet POSIX noch das Setzen des bereits erwähnten 'Sticky Bits'. Dieses ermöglicht es, Verzeichnisse und Dateien vor Umbenennung und Löschung durch Nicht-Besitzer - unabhängig von den restlichen Rechten oder der Gruppenzugehörigkeit - zu schützen. Auch die SUID- & SGID-Bits bieten etwas zusätzliche Flexibilität. So ist es möglich, normalen Benutzern das Ausführen eines 'Scripts' - welches bspw. root-Rechte benötigt - zu erlauben. Das Script/Programm wird dann unter der UID bzw. GID des Besitzers bzw. der Gruppe ausgeführt. Der Begriff 'Scripts' steht in Anführungszeichen, da auf den meisten Unix Systemen das SUID-Bit bei nicht-Binärprogrammen aus Sicherheitsgründen ignoriert wird. Dementsprechend müssen kompilierte Programme für solche Operationen erzeugt werden. Da mittels VIFS die RNP-Infrastruktur auf ein POSIX-konformes System abgebildet werden soll, können die von der POSIX-Schnittstelle zur Verfügung gestellten Eigenschaften vollständig genutzt werden.

Mengendefinitionen

Um ein POSIX-Dateisystem kompakt zu beschreiben, folgt die Definition der benötigten Mengen. Die hierbei entstehenden Mengen dienen zur Darstellung der existierenden Objekte auf einem einzelnen POSIX-basierten System. Dementsprechend gibt es für verschiedene Systeme verschiedene Mengen, welche anhand des tiefgestellten Index unterschieden werden.

Hilfsmengen

$i, j \in N$

- Nutzer:
  Sei U_i die Menge der Nutzer auf System i. Ein Element dieser Menge u^j besitzt eine eindeutige Nutzer-ID (UID), welche an dieser Stelle durch den Index repräsentiert werden kann.

- Gruppen:
  Sei G_i die Menge der POSIX-Gruppen auf System i. Ein Element dieser Menge g^j besitzt eine eindeutige Gruppen-ID (GID) sowie eine endliche Anzahl von Elementen der Menge U, welche die Gruppenzugehörigkeit diverser Nutzer beschreibt.
  $G_\mathrm{i} := \{(GID, \{u\}) \mid \ GID \in N,\ u \in U_\mathrm{i}\}$

- Rechte:
  Sei A die Menge der möglichen Konstellationen von POSIX-Rechten.
  $A := \{((abc), (def), (ghi)) \mid$
  $a, d, g \in \{r, -\},\ b, e, f \in \{w, -\},\ c, f \in \{x, s, S, -\},\ i \in \{x, t, T, -\}\}$
  Das erste Tripel beschreibt die Rechte für den Nutzer, das zweite Tripel die Rechte für die Gruppe und das dritte Tripel beschreibt die Zugriffsrechte für alle Nutzer, die nicht durch Besitzer oder Gruppe abgedeckt sind. Ein Element a^{i} entspricht dahingehend einer bestimmten Kombination von Zugriffsrechten.

- Dateien und Verzeichnisse:
  Sei F_{i} die Menge der auf System i vorhandenden Dateien und Verzeichnisse. Ein Element f^{j} besteht aus dem eindeutigen Pfad der Datei oder des Verzeichnisses im POSIX-Dateisystem.

POSIX-Dateisystem

$i, j \in N$

Sei V die Menge verschiedener POSIX-Dateisysteme und v_{i} ein Element dieser Menge.
$V := \{(\{(f, u, g, a)\}) \mid\ f \in F_{\mathrm{i}},\ u \in U_{\mathrm{i}},\ g \in G_{\mathrm{i}},\ a \in A\}$
Ein solches Element v_{i} beschreibt ein einzelnes POSIX-Dateisystem samt aller darin existierenden Dateien sowie Verzeichnisse und den zugehörigen Rechten für die ebenfalls demselben POSIX-Dateisystem zugehörigen Nutzer und Gruppen.

2.4 OSI-Management

Um im folgenden Szenario eine passende Domänenunterteilung zu treffen, bietet es sich an, das OSI-Organisationkonzept ([HGH]) zu verwenden. Dieses Konzept unterscheidet zwischen Verwaltungs- und Organisationsdomänen. Organisationsdomänen werden nach funktionellen Gesichtspunkten gebildet, Verwaltungsdomänen sind zuständig für die Erstellung und Manipulation von Organisationsdomänen. Innerhalb einer Verwaltungsdomäne unterstehen grundsätzlich alle Managementobjekte genau einer Verwaltungsautorität. Als Managementobjekte können Rechner, VM, physische wie auch virtuelle Schnittstellen, Benutzer aber auch Rechte und weitere Domänengruppierungen betrachtet werden. Durch die Verschachtelung virtueller Systeme ist es naheliegend auch Organisationsbereiche zu verschachteln und so Verwaltungsdomänen als Managementobjekte von übergeordneten Verwaltungsdomänen zu betrachten. Hierbei ist nicht ausgeschlossen, dass Teilnehmer einer Verwaltungsdomäne ebenfalls Mitglieder von einer oder mehreren Organisationsdomänen sind.

Mengendefinitionen

Um im weiteren Vorgehen die abzubildende Infrastruktur in einer greifbaren Form darstellen zu können, werden in diesem Kapitel die benötigten Mengen beschrieben. Geht man von einer OSI-Management Struktur aus, werden hierbei zwei Mengen zur Darstellung der Domänen benötigt. Um diese Mengen zu erzeugen werden weitere Hilfsmengen benötigt.

Hilfsmengen

$i, j \in N$

- Personen:
  Sei P die Menge der Personen mit Zugriff auf die Infrastruktur. Ein Element dieser Menge p^{i} könnte hierbei aus den im Unternehmen vorhandenen Informationen zur Person bestehen. An dieser Stelle ist jedoch lediglich eine eindeutige ID zwingend, welche durch den Index dargestellt werden kann.

- Komponenten:
  Sei K die Menge der in der Infrastruktur existierenden Komponenten und k^{i} ein Element dieser Menge. k^{i} benötigt ebenfalls eine eindeutige ID und besitzt zusätzlich die endliche Menge der von der Komponente zur Verfügung gestellten Operationen $O_{\mathrm{k^i}}$. Ein Element o^{j} der Menge $O_{\mathrm{k^i}}$ stellt eine eindeutige auf der Komponente k^{i} ausführbare Operation dar.

  $K := \{(id, O_{\mathrm{k}}) \mid id \in N\}$

  Wird in weiterer Folge von ${k^{\mathrm{i}}}_{+}$ gesprochen, handelt es sich um eine bestimmte Komponente der Infrastruktur samt aller zur Verfügung gestellten Operationen. Der Term ${k^{\mathrm{i}}}_{-}$ repräsentiert dahingehend eine bestimmte Komponente der Infrastruktur, welche jedoch nicht alle zur Verfügung gestellten Operationen beinhaltet. Weiters könnte ${k^{\mathrm{i}}}_{-}$ genauer definiert werden durch die Auflistung der entsprechenden Operationen: ${k^{\mathrm{i}}}_{-} := (id, \{o^{\mathrm{j}}, ...\})$

Domänen

Mithilfe der zuvor erzeugten Hilfsmengen ist die Erstellung der benötigten Domänen in Mengennotation möglich.

$i, j \in N$

- Organisationsdomänen:
  Sei D_{o} die Menge der in der Infrastruktur existierenden Organisationsdomänen:
  $D_{\mathrm{o}} := \{(\{p\}, \{{k_{-}}^{\mathrm{i}}\}) \mid p \in P,\ k \in K\}$
  Ein Element der Menge D_{o} (${d_{\mathrm{o}}}^{\mathrm{j}}$) repräsentiert eine einzelne Organisationsdomäne. Da ${d_{\mathrm{o}}}^{\mathrm{j}}$ wiederum Mengen beinhaltet, wird ein Element der in ${d_{\mathrm{o}}}^{\mathrm{j}}$ enthaltenen Komponentenmenge mit ${d_{\mathrm{o}}}^{\mathrm{j^i}}$ identifiziert und repräsentiert die Zugriffsrechte von - der Domäne ${d_{\mathrm{o}}}^{\mathrm{j}}$ angehörigen - Personen auf die in ${k_{-}}^{\mathrm{i}}$ enthaltenen Operationen. Welche Operationen die genannten Personen besitzen ist zu diesem Zeitpunkt nicht von Belangen. Einzig relevant ist, dass die Zugriffsrechte der entsprechenden Personen beschränkt sind.

- Verwaltungsdomänen:
  Sei D_{v} die Menge der in der Infrastruktur existierenden Verwaltungsdomänen:
  $D_{\mathrm{v}} := \{(\{p\}, \{{k_{+}}^{\mathrm{i}}\}) \mid p \in P,\ k \in K\}$
  Ein Element der Menge D_{v} (${d_{\mathrm{v}}}^{\mathrm{j}}$) repräsentiert eine einzelne Verwaltungsdomäne. Da Verwaltungsdomänen im Stande sein müssen Organisationsdomänen zu erzeugen, also Personen Operationen zu erlauben, wird davon ausgegangen, dass Mitglieder dieser

Domäne alle Operationen auf den ihnen zugewiesenen Komponenten ausführen können. Dementsprechend werden durch $d_v{}^{j^i}$ Komponenten angesprochen, für die die Personen der Verwaltungsdomäne zuständig sind und Vollzugriff darauf besitzen.

Mittels der definierten Mengen ist man nun in der Lage, die OSI-Domänen-Struktur jeder beliebigen Infrastruktur darzustellen.

Die in diesem Kapitel definierten Mengen werden in weiterer Folge zur mathematischen Abbildung der, dem folgenden Szenario zugrundeliegenden, Infrastruktur in das VIFS-Dateisystem benötigt.

3 Szenario

Als Szenario für die Analyse der Anforderungen dieser Arbeit dient das Rechnernetzepraktikum(RNP) der LMU. Das RNP dient Studenten dazu, ihre Fertigkeiten im Bereich Netzmanagement zu festigen und detaillierte Abläufe verschiedener Protokolle und Konfigurationsvorgänge zu verstehen und anzuwenden. Dabei steht den Studenten eine geschachtelte virtuelle Infrastruktur zu Verfügung, auf der sie Aufgaben wie beispielsweise die Umsetzung verschiedener Routingprotokolle, das Einrichten von IP-Tunneling oder das Konfigurieren von DNS-Servern bewältigen müssen. Weitere Details über konkret behandelte Themen sowie den Ablauf des Praktikums findet der interessierte Leser unter [rnp].

3.1 Aufbau des RNP

Um den Studenten eine umfangreiche Testumgebung zu bieten, werden virtuelle Maschinen(VM) - im Folgenden als Labor bezeichnet - verwendet, welche auf mehreren physischen Maschinen betrieben werden. Ein solches Labor ist ein Host für neun weitere virtuelle Maschinen und trägt den entsprechenden Gruppennamen als Bezeichner. Die vom Labor gehosteten und betriebenen 2-VM namens router01 bis router04, pc01 bis pc03, sowie client und edge werden von nun an als Laborumgebung bezeichnet.

Ein auszugsweiser Aufbau der Labore ist in Abbildung 3.1 zu sehen. Die getrennten Segmente beschreiben das jeweilige Virtualisierungslevel, die Pfade zwischen den Elementen die 'Client-Host'-Beziehungen. An dieser Stelle soll erwähnt sein, dass i+1-VM stets innerhalb einer i-VM betrieben werden. Der Vollzugriff für Studenten beschränkt sich ausschließlich auf die Elemente der Laborumgebung, in welcher die gestellten Aufgaben gelöst werden müssen. Alle Elemente verfügen über eine Netzanbindung, um den Studenten Remote-Zugriffe zu ermöglichen. Diese Netzanbindung erfolgt stets über die virtuelle eth0-Schnittstelle der jeweiligen 2-VM. Des Weiteren verfügen alle Laborumgebungselemente über mehrere virtuelle Ethernet-Schnittstellen um die laborumgebungsinternen Maschinen zu verbinden. Die Vernetzung zwischen den 2-VM, ohne explizite Darstellung der benötigten zwischengeschalteten virtuellen Switches, ist Abbildung 3.2 zu entnehmen. Um diesen Netzplan der virtualisierten Umgebung zu realisieren, werden virtuelle Switches eingesetzt, welche von der zuständigen 1-VM erzeugt und verwaltet werden müssen. Eine Auflistung und Zuordnung von virtuellen Switches zu virtuellen Schnittstellen, welche für jede Laborumgebung betrieben werden muss, liegt in Tabelle 3.1 vor. Einen detaillierteren Einblick in die Gesamtstruktur inklusive benötigter virtueller Switches bietet [Dan16a].

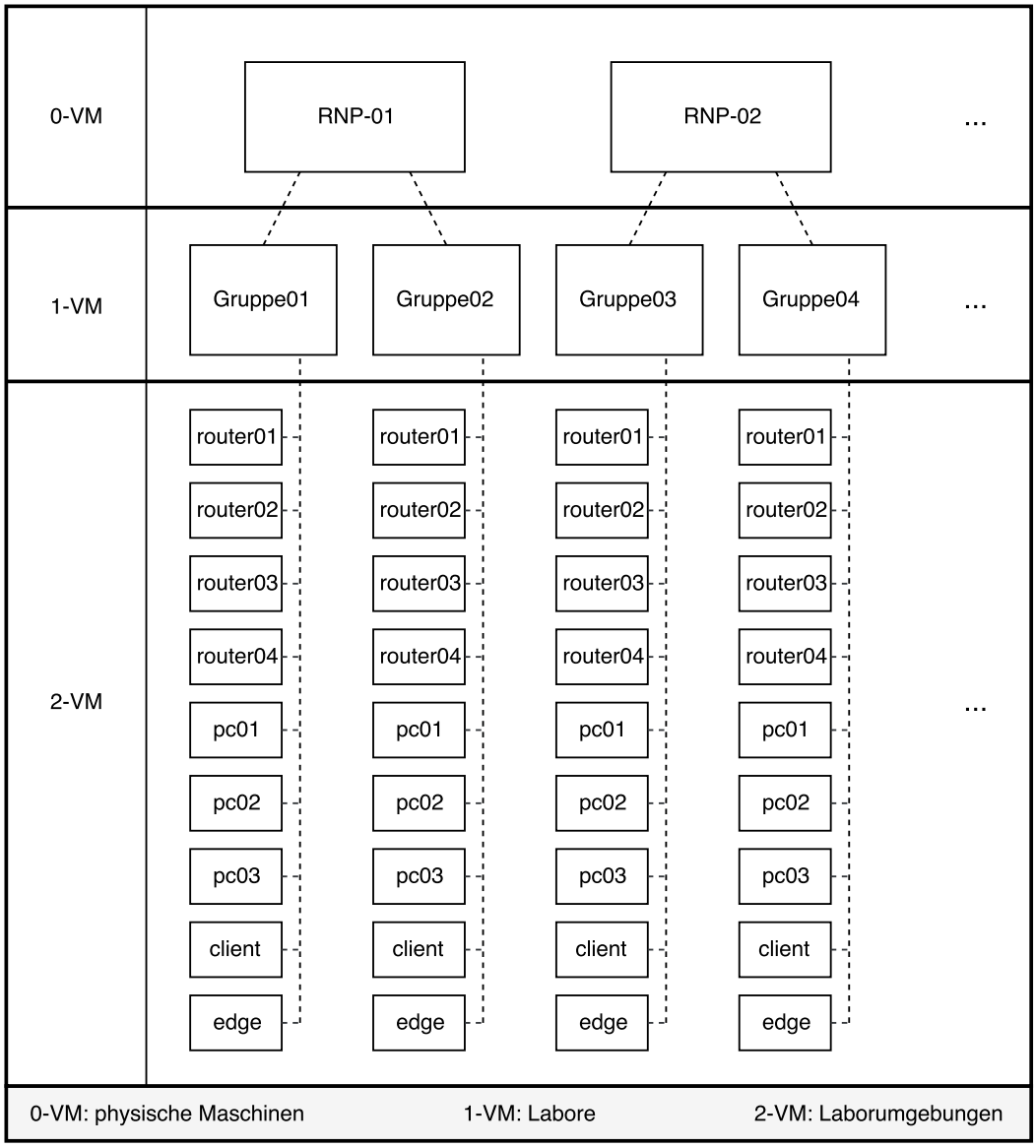

Abbildung 3.1: Aufbau der RNP-Infrastruktur

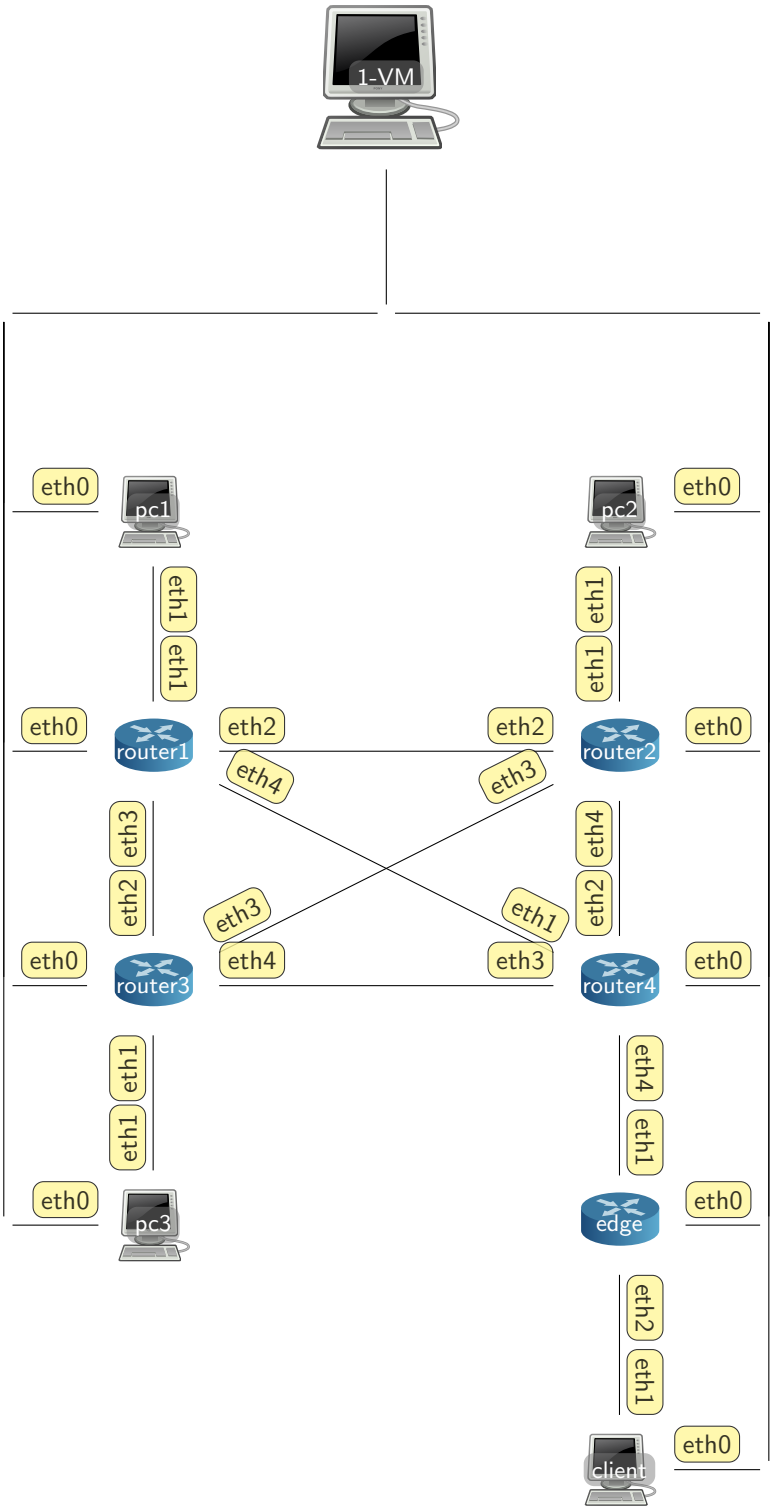

Abbildung 3.2: Netzplan einer Laborumgebung

Tabelle 3.1: Switches und damit verbundene Schnittstellen

Switch auf 1-VM	Schnittstelle 1	Schnittstelle 2
Switch1	pc1.eth1	router01.eth1
Switch2	pc2.eth1	router02.eth1
Switch3	pc3.eth1	router03.eth1
Switch4	edge.eth1	router04.eth4
Switch5	edge.eth2	client.eth1
Switch6	router01.eth0	GruppeXX.vmbr0
	router02.eth0	
	router03.eth0	
	router04.eth0	
	pc1.eth0	
	pc2.eth0	
	pc3.eth0	
	edge.eth0	
	client.eth0	
Switch12	router01.eth2	router02.eth2
Switch13	router01.eth3	router03.eth2
Switch14	router01.eth4	router04.eth1
Switch23	router02.eth3	router03.eth3
Switch24	router02.eth4	router04.eth2
Switch34	router03.eth4	router04.eth3

3.2 RNP-Domänen

Das Praktikum ist für 10 Teilnehmergruppen vorgesehen, wobei eine Teilnehmergruppe zu Beginn zwei Studenten beinhaltet. Natürlich kommt es vor, dass während des Semesters Teilnehmer vom Praktikum abspringen. Um die verbliebenen Studenten vor der alleinigen Bearbeitung der relativ umfangreichen Aufgaben zu bewahren, werden Gruppen zusammengefasst. Es ist also durchaus möglich, dass eine Teilnehmergruppe im Verlauf des Semesters drei oder mehr Studenten umfasst. Des Weiteren verfügt das RNP über einen das Praktikum leitenden Administrator, einen Praktikumsbetreuer sowie zwei bis vier Tutoren. Eine namentliche Zuordnung von Personen zu Aufgabenbereichen/Rollen ist in Tabelle 3.2 ersichtlich. Der Aufgabenbereich der Tutoren kann hierbei von Tutor zu Tutor variieren und ist somit nicht strikt festgelegt. Ein langjähriger Tutor des RNP könnte beispielsweise auch bestimmte Aufgaben des RNP-Betreuers mit übernehmen, wohingegen Tutoren, welche erstmalig das RNP unterstützen, mit diesen Aufgaben vermutlich überfordert wären.

Um den zugewiesenen Aufgabenbereichen nachzukommen, benötigt jede Person ein Benutzerkonto pro Maschine. Den Studenten wird in der Praxis der Zugriff auf die Laborumgebungsmaschinen über das root-Konto gewährt. Um jedoch ein klareres und leichter verständliches Domänenbild vermitteln zu können, wird hierbei davon ausgegangen, dass jeder Student ein Benutzerkonto (studentXY) für den Login erhält. Der Zugriff auf die Laborumgebung ist nur vom zugehörigen Labor aus möglich. Dahingehend benötigen die Studenten also zusätzlich noch Benutzerkonten auf den ihnen zugeteilten Laboren. Die Zuordnung von Personen zu Benutzerkonten ist in Tabelle 3.3 zu sehen.

Tabelle 3.2: Personen und Aufgabenbereiche

Name	Rolle
RNP-Admin	Administrator auf allen physischen RNP-Maschinen
Praktikumsbetreuer	Administrator auf allen physischen RNP-Maschinen Administator auf allen 1-VM
Tutor01	Administator auf zugewiesenen 1-VM, Nutzer/Admin auf zugewiesenen physischen RNP-Maschinen
Tutor02	Administator auf zugewiesenen 1-VM, Nutzer/Admin auf zugewiesenen physischen RNP-Maschinen
Student01	Administator auf allen Maschinen der Laborumgebung Gruppe01, Nutzer auf Gruppe01
Student02	Administator auf allen Maschinen der Laborumgebung Gruppe01, Nutzer auf Gruppe01
Student03	Administator auf allen Maschinen der Laborumgebung Gruppe02, Nutzer auf Gruppe02
Student04	Administator auf allen Maschinen der Laborumgebung Gruppe02, Nutzer auf Gruppe02
...	...

Tabelle 3.3: Personen und Benutzerkonten

Name	Maschine	Kontoname	Kontotyp
RNP-Admin	RNP01	admin	Administrator
	RNP02	admin	Administrator
	...	...	...
Praktikumsbetreuer	RNP01	betreuer	Administrator
	RNP02	betreuer	Administrator
	...	...	...
	Gruppe01	betreuer	Administrator
	Gruppe02	betreuer	Administrator
	...	...	...
Tutor01	Gruppe01	tutor01	Administrator
	...	...	...
Tutor02	Gruppe02	tutor02	Administrator
	...	...	...
Student01	Gruppe01	student01	Benutzer
	router01 (Gruppe01)	student01	Administrator
	...	...	...
Student02	Gruppe01	student02	Benutzer
	router01 (Gruppe01)	student02	Administrator
	...	...	...
Student03	Gruppe02	student03	Benutzer
	router01 (Gruppe02)	student03	Administrator
	...	...	...
Student04	Gruppe02	student04	Benutzer
	router01 (Gruppe02)	student04	Administrator
	...	...	...
...	...	...	...

Um die existierenden Domänen aufzuzeigen, wird auf das in Kapitel 2.4 erläuterte OSI-Domänenkonzept Bezug genommen. Aufgrund der verhältnismäßig geringen Ausmaße des

RNP, kann von einer einzelnen Verwaltungsdomäne ausgegangen werden. Es ist naheliegend, den RNP-Leiter und den Praktikumsbetreuer als Verwaltungsauthorität zu benennen. Möchte man die existierenden Organisationsdomänen darstellen, müssen die Gebiete zusammengefasst werden, für die eine entsprechende Gruppierung oder gegebenenfalls auch eine Einzelperson verantwortlich ist. Um dieser Verantwortung nachzukommen, ist es allerdings oft notwendig, den entsprechenden Verantwortlichen auch außerhalb ihres Verantwortungsbereichs Zugriffsrechte einzuräumen. So werden den Studenten auf den zugehörigen Laboren (1-VM) Rechte eingeräumt, um bestimmte Features der Virtualisierungssoftware zu nutzen und so Teile ihrer Laborumgebung (beispielsweise nach einer Fehlkonfiguration von eth0) selbst neu starten oder zurücksetzen zu können. Auch der RNP-Betreuer muss das Management der Gruppen-VM teilweise vom hostenden RNP-Rechner aus durchführen. Daher wird kurz auf eine mögliche Unterscheidung der Zugriffe auf Operationen eingegangen.

Externe Zugriffe: Dies betrifft bei Einzelrechnern oder auch bei verteilten Systemen beispielsweise Stromzufuhr, Betriebsort, Kühlung, RAM-Menge usw. Bei einer virtualisierten Maschine wären dies alle Operationen, die vom Host am Gast mittels Virtualisierungssoftware (xen, kvm, etc) vorgenommen werden können, wie beispielsweise Migration, Löschen oder Erstellen einer VM, Erhöhung/Verringerung von Speichermenge oder CPU-Zeit an bestehenden VM.

Interne Zugriffe: Dies betrifft alle Operationen, die auf dem betreffenden System direkt am Betriebssystem ausgeführt werden können, wie beispielsweise Rechtemanagement, Netzkonfiguration, Herunterfahren und Neustarten des Systems.

Konfiguriert Student01 also beispielsweise die Schnittstelle eth4 auf router01 mittels eines Benutzerkontos auf router01, handelt es sich um einen internen Zugriff. Startet Student01 router01 mittels des Befehls “xl restart router01” vom Labor Gruppe01 aus neu, handelt es sich um einen externen Zugriff an router01. Die durch diese Zugriffsunterteilung entstehenden Möglichkeiten zur weiteren Domänenunterteilung sind in Abbildung 3.3 zu sehen. Eine konkrete Domänenzuteilung kann also, je nach gewünschtem Detailgrad, bereits erheblichen Aufwand verursachen. Abbildung 3.4 zeigt eine einfache Darstellung (keine zusätzliche Beachtung von externen und internen Zugriffen) der Tutoren- und Studentendomänen. Abbildung 3.5 zeigt mögliche Domänengrenzen unter Beachtung von externen und internen Zugriffen. Hierbei Überschneiden sich die Zugriffsbereiche von Studenten und Tutoren sobald den Studenten die Benutzung von Operationen die als externer Zugriff eingeordnet werden können gewährt werden. In anderen Strukturen könnte so eine Situation durch die gemeinsame Nutzung von Komponenten, beispielsweise Druckern oder Speichermedien, zustande kommen.

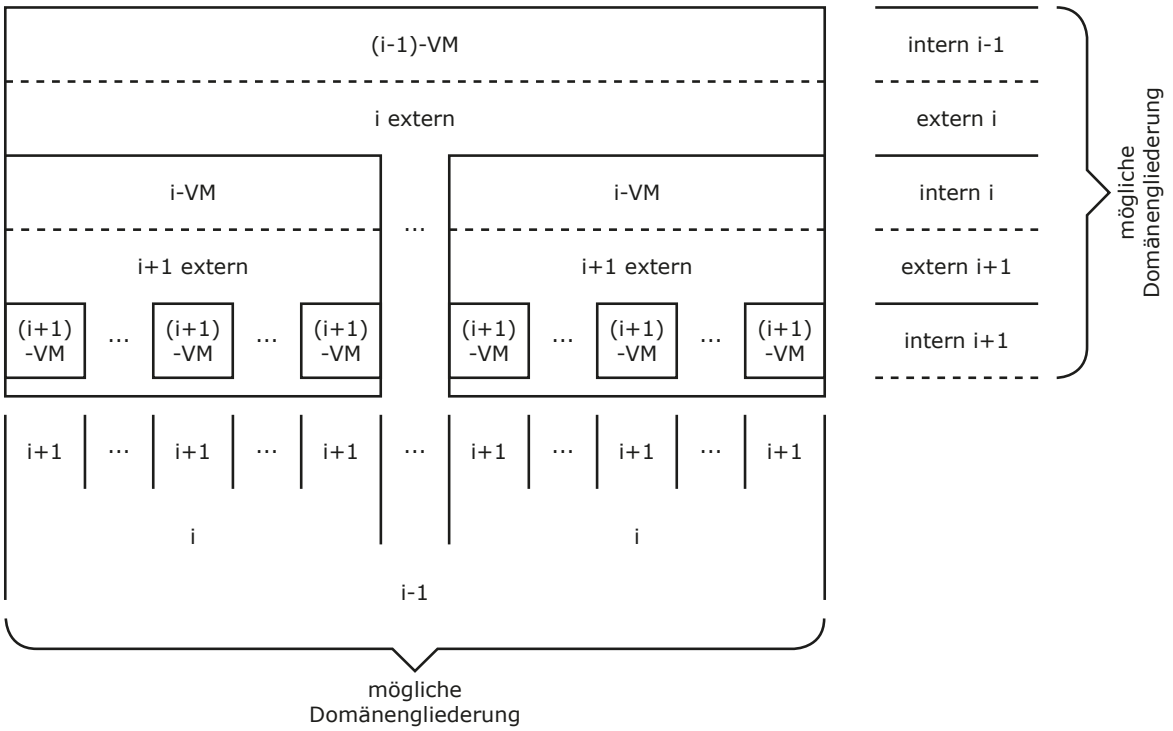

Abbildung 3.3: Mögliche Domänengliederung durch Trennung von externen & internen Zugriffen

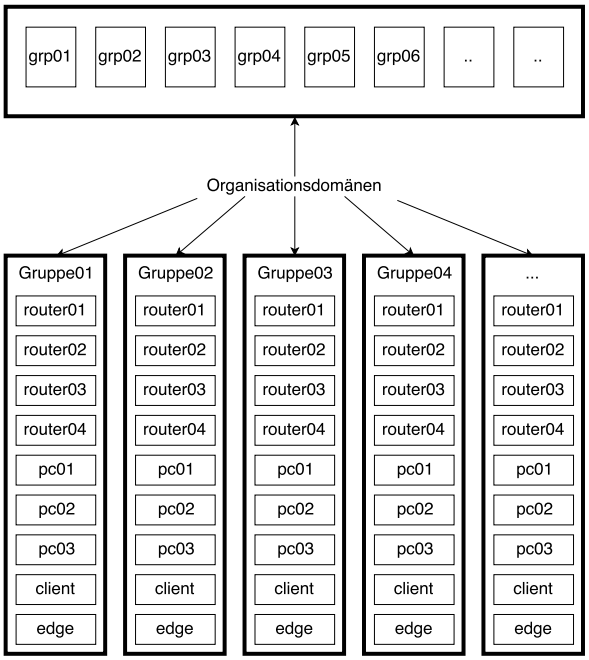

Abbildung 3.4: Möglicher Organisationsdomänenauszug des RNP

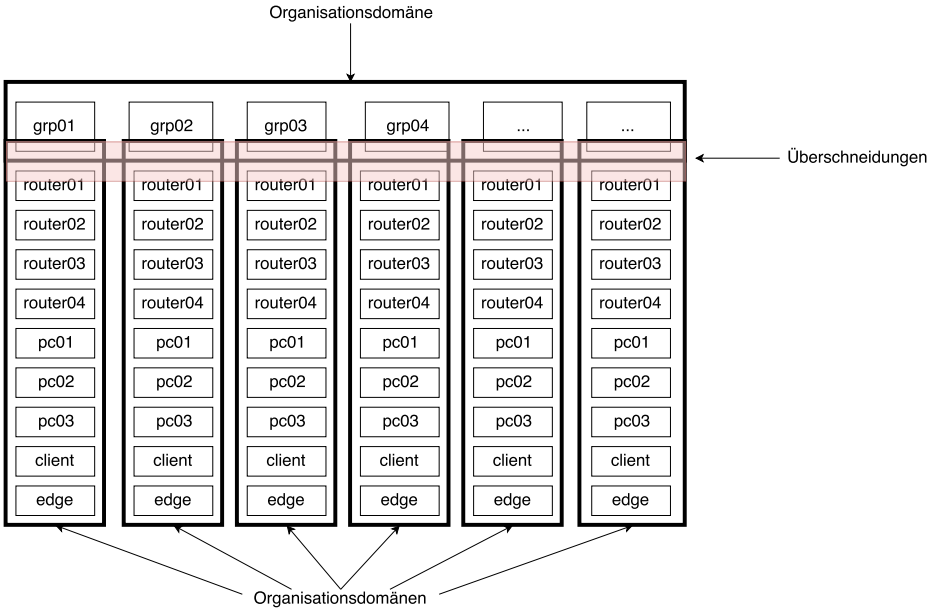

Abbildung 3.5: Möglicher Organisationsdomänenauszug des RNP bei Beachtung von internen & externen Zugriffen

3.3 RNP-Operationen

Jede Domänengruppierung hat ihre Aufgaben. So liegt es im Aufgabenbereich des RNP-Admins und des RNP-Betreuers die Funktionalität der physischen Maschinen RNP-XX zu gewährleisten. Dazu gehören physische Interaktionen wie eine Netzverbindung unter den Maschinen herzustellen, das Konfigurieren der Sicherheitseinstellungen auf den Betriebssystemen der Geräte, die indirekte Wartung und Konfiguration der gehosteten 1-VM mittels der zur Verfügung gestellten Operationen der Virtualisierungssoftware, das Erstellen und Konfigurieren virtueller Switches uvm. Der Aufgabenbereich des RNP-Betreuers und evtl. der Tutoren umfasst das Sicherstellen der benötigten Funktionalität der Labore (1-VM). Hierzu gehört das Pflegen der virtuellen Netzumgebung sowie die Erstellung der von den Studenten benötigten Benutzerkonten und des Weiteren auch die indirekte Wartung der 2-VM mittels der von der Virtualisierungssoftware zur Verfügung gestellten Operationen. Konfigurieren Studenten die eth0-Verbindung einer Maschine aus ihrer Laborumgebung fälschlicherweise und haben somit keinen Zugriff mehr auf die entsprechende Maschine, liegt es an der übergeordneten Domäne, die betreffende Maschine neu zu erstellen oder Konfigurationen an dieser vorzunehmen. Eine weitere häufig vorkommende Lösung für die genannte Situation ist es, den Studenten den Zugriff auf die benötigten Operationen zu erlauben. Den Domänen zugeordnete Operationen sind in den Tabellen 3.4, 3.5 und 3.6 aufgelistet und den Bereichen des FCAPS-Schemas ([IG]) zugeordnet.

Im vorherigen Unterkapitel wurde bereits eine mögliche Unterteilung von Domänen durch interne und externe Zugriffe angesprochen. Dementsprechend müssen auch die Operationen diesen Unterteilungen zugeordnet werden. Hierfür muss zuvor jedoch eine Analyse der betroffenen Operationen stattfinden. Der Großteil der externen Operationen an einer physischen Maschine bedarf keiner weiteren Betrachtung. Das Einstecken eines Netzkabels in den

Tabelle 3.4: Operationsauszug von Studenten auf Laborumgebungsmaschinen

Bereich	Operation
Fehlermanagement	Beheben von falsch konfigurierten Programmen wie bind9, dhcp, ip oder Schnittstellen (eth1-eth4), sowie das Finden von Fehlern in der Laborumgebung
Konfigurationsmanagement	Konfigurieren von neu installierten Programmen oder Schnittstellen
Benutzerkontenmanagement	Anlegen eines neuen Benutzerkontos pro Student und Hinzufügen dieses Kontos zur sudo-Gruppe
Leistungsmanagement	Überwachen der Dauer (in ms) von Dateitransporten bei verschiedener Wegewahl
Sicherheitsmanagement	Sicherstellen der sicheren Verwendung von Programmen, bspw. SNMP
Bereichsübergreifende Operationen	Neustarten, Herunterfahren, Schnittstellen deaktivieren/aktivieren, Backups anlegen und einspielen

Tabelle 3.5: Operationsauszug von Tutoren auf Laboren

Bereich	Operation
Fehlermanagement	Beheben von Fehlfunktionen an den Gruppen-VM
Konfigurationsmanagement	Konfigurieren der Labore und der virtuellen Netze
Benutzerkontenmanagement	Anlegen von Benutzerkonten für Studenten, Löschen von Benutzerkonten von nicht mehr teilnehmenden Studenten
Leistungsmanagement	Überwachen des Leistungsbedarfs der Labore bzw. Laborumgebungen, Vergeben von Kontingenten bei Überbeanspruchung einzelner Gruppen, Deaktivieren nicht mehr benötigter Labore
Sicherheitsmanagement	Unterbinden von Hackversuchen übermütiger Studenten
Bereichsübergreifende Operationen	Neustarten, Herunterfahren, Backups anlegen und einspielen, Zurücksetzen von Teilen der Laborumgebungen

Tabelle 3.6: Operationsauszug von RNP-Admin/-Betreuer auf RNP-Maschinen

Bereich	Operation
Fehlermanagement	Beheben von Fehlfunktionen
Konfigurationsmanagement	Konfigurieren der Labore und der virt. & phys. Netze
Benutzerkontenmanagement	Anlegen von Benutzerkonten für Tutoren, Löschen von Benutzerkonten von nicht mehr teilnehmenden Tutoren, *Delegation von Zugriffsrechten*
Leistungsmanagement	Überwachen des Leistungsbedarfs der Gruppen-VM bzw. Vergeben von Kontigenten bei Überbeanspruchung einzelner Gruppen, Migration von VM Deaktivieren nicht mehr benötigter Labore
Sicherheitsmanagement	Betreiben der Firewalls
Bereichsübergreifende Operationen	Neustarten, Herunterfahren, Backups anlegen und einspielen, Zurücksetzen von Laboren

dafür vorgesehenen Schnittstellen-Port kann eindeutig als externe Operation erkannt werden. Jedoch darf hierbei nicht vergessen werden, dass aufgrund dieser Operation die meisten Betriebssysteme automatische Operationen veranlassen (beispielsweise das Aktivieren der Schnittstelle und des DHCP-Clients), welche zur korrekten Konfiguration der Schnittstelle führen. Bei Heimanwender-PCs führt das Drücken des Power-Buttons - sofern das darauf laufende Betriebssystem diese Aktion unterstützt - dazu, dass das Betriebssystem heruntergefahren oder in den Ruhezustand versetzt wird. Bei einer VM verhält es sich genauso. Das Ausführen eines durch Virtualisierungssoftware zur Verfügung gestellten Befehls zum Herunterfahren oder Neustarten der VM vom Host aus, veranlasst das Betriebssystem der VM die verlangte Aktion auszuführen. Befindet sich das Betriebssystem der VM in einem kritischen Zustand (bspw. durch einen das System blockierenden Deadlock), helfen in den meisten Fällen auch die externen - das Zielsystem schonende Operationen (reboot/shutdown) - vom Host aus nicht mehr. Somit könnte durchaus begründet werden, dass eine externe Operation einen Auslöser für eine interne Operation auf der betreffenden Maschine darstellt. Das ist jedoch nicht überall der Fall. Wird eine VM beispielsweise auf einen anderen Host migriert, werden hierbei keine internen Operationen der betreffenden VM, sondern interne Operationen des neuen Hosts ausgeführt. Dennoch stellt die Migration einer VM eine externe Operation an der VM dar, genauso wie das Ändern des Betriebsortes eines physischen Rechners. Eine detaillierte Zuordnung von externen und internen Operationen wird in Tabelle 3.7 aufgezeigt.

Tabelle 3.7: Auszug von internen & externen Operationen

Operation	initiiert durch/auf	betroffen	Zuordnung
Herunterfahren/Neustarten	router01	router01	intern
Herunterfahren/Neustarten	Gruppe01	Gruppe01	intern
		router01	extern
		router02	extern
		...	extern
Herunterfahren/Neustarten	RNP-01	RNP-01	intern
		Gruppe01	extern
		router01	extern
		...	extern
		Gruppe02	extern
		router01	extern
		...	extern
Herunterfahren/Neustarten	(Stromzufuhr)	RNP-01	extern
		Gruppe01	extern
		router01	extern
		...	extern
		Gruppe02	extern
		router01	extern
		...	extern
Ändern RAM-Menge	Gruppe01	router01	extern
Ändern RAM-Menge	RNP-01	Gruppe01	extern
Hinzufügen virt. Schnittstellen	Gruppe01	router01	extern
Hinzufügen virt. Schnittstellen	RNP-01	Gruppe01	extern
Hinzufügen phys. Schnittstellen	(Techniker)	RNP-01	extern
Migration router01 nach Gruppe02	Gruppe01	Gruppe02	extern
		router01	extern
Migration Gruppe01 nach RNP-02	RNP-01	RNP-02	extern
		Gruppe01	extern
Ändern von IP-Adressen	router01	router01	intern
(De)-Aktivieren virt. Schnittstellen	router01	router01	intern
Reload des DNS-Servers	router01	router01	intern
Lesen des Syslog	router01	router01	intern
Lesen des Syslog	Gruppe01	Gruppe01	intern
Erstellen eines virt. Switches zw. router01 & router02	Gruppe01	router01	extern
		router02	extern
Delegation des Rechts der Ausführung eines Scripts auf Gruppe01 an student01	RNP-Betreuer/Gruppe01	Gruppe01	intern

3.4 Außerordentliche Use-Cases

Wie bereits in Kapitel 3.2 erwähnt, können außerordentliche Situationen die Domänenstruktur und Aufgabenbereiche von mitwirkenden Personen beinflussen. So geschieht es am häufigsten, dass Studenten sich erst in der Mitte des Semesters dem Aufwand des Praktikums bewusst werden und sich abmelden. In einem solchen Fall muss die Leitung des RNP Maßnahmen zur Zusammenführung von Gruppen treffen, um den verbliebenen Studenten vor der aufwändigen alleinigen Bearbeitung der Aufgaben zu bewahren. Ist bereits klar, welche Gruppen zusammengelegt werden, muss der Nutzer, dessen Arbeitsumfeld geändert wird, Zugriffsberechtigungen auf die neue Umgebung erhalten und in etwaige Gruppen hinzugefügt werden. Da das RNP aktuell keine zentrale Benutzer- und Gruppenverwaltung verwendet, muss der Nutzer auf allen Laborumgebungselementen sowie auf dem zugehörigen Labor

angelegt und mit Rechten ausgestattet werden. Dieser Aufwand kann bei mehrmaligem Vorkommen dieser Situation durchaus beträchtliche Ausmaße erreichen.

Ein weiteres mögliches außerordentliches Szenario könnte bei einer längerfristigen unvorhersehbaren Erkrankung von Administrationspersonal entstehen. Fällt beispielsweise der Betreuer auf unbestimmte Zeit aus, muss eine Vertretung bestimmt und mit benötigten Nutzern und Rechten ausgestattet werden. Selbes gilt für die Tutoren.

Ein unwahrscheinlicher aber dennoch möglicher Fall wäre die Förderung von außergewöhnlich engagiertem Studentenverhalten. Hierbei könnte das Management eines Labors zur Steigerung der durch das RNP errungenen Kenntnisse des Studenten in dessen Hand gelegt werden. Jedoch würde auch diese Situation einen nicht zu vernachlässigenden Verwaltungsaufwand bei der Gruppenzuteilung und der Vergabe von Zugriffsrechten erzeugen. Schließlich soll der entsprechende Student lediglich für das Management seiner Umgebung und nicht der gesamten RNP-Struktur Zugriffsrechte erhalten.

Die aufgezeigten Use-Cases stellen lediglich einen Auszug dar und so muss mit vermehrtem Auftreten solcher Situationen gerechnet werden. Hierbei wird sehr gut ersichtlich, dass die benötigte Rechteverwaltung ein ausreichendes Maß an Flexibilität benötigt.

4 Anforderungsanalyse

Durch die VIFS-Abbildung eines Managementsystems in ein Dateisystem, wird das Betreten eines Verzeichnisses gleichgesetzt mit dem Überschreiten einer Systemgrenze und das Schreiben in eine Datei zur Ausführung einer Management-Operation. Die Vereinfachung dieser Prozesse führt allerdings zu einer Erschwerung der Sicherheitspolitik. Um im Vorfeld alle potenziellen Problemfälle, die auf das Rechtemodell einwirken könnten, zu erkennen, folgt in Kapitel 4.1 eine Analyse der benötigten Management-Operationen innerhalb eines Managementsystems sowie die Beschreibung einer generischen Schnittstelle. Diese Schnittstelle wird benötigt, um alle Anforderungen seitens der Operationen an das Rechtemodell klar zu definieren. Kapitel 4.2 widmet sich den aus der Schnittstelle sowie dem Szenario entstehenden Anforderungen an das Rechtemodell.

4.1 Managementoperationen und API

Um die notwendigen Operationen eines Managementsystems zu eruieren, ist es zuerst notwendig einen Überblick über dessen mögliche Bestandteile zu erlangen. Aus dem RNP ergeben sich bereits einige davon. Ein Router bietet andere Funktionalität als ein Switch. Eine virtuelle Maschine stellt ebenfalls andere Operationen zur Verfügung als eine physische Maschine. Nutzer unterscheiden sich durch erteilte Rechte, Rollen dienen zur Gruppierung von Nutzern mit selbem Aufgabenbereich und somit denselben Rechten. Denkt man an größere Infrastrukturen können weitere unterschiedliche Komponenten wie beispielsweise zusätzliche Speichermedien, Drucker oder auch Gruppierungskomponenten wie komplex verschachtelte Domänen mitwirken. Minimiert man jedoch, unabhängig von den Ausmaßen der vorliegenden Strukturen, die Bestandteile auf die kleinsten gemeinsamen Nenner, so erhält man die Folgenden:

- Nutzer
  Ein Nutzer gehört stets einer Domäne an. Des Weiteren verfügt er über eine Sammlung von Zugriffsrechten, welche ihm die Ausführung verschiedener Operationen erlauben. Nutzer benötigen keine weitere Unterteilung, da sie sich lediglich durch die ID sowie die zugewiesenen Rechte unterscheiden.

- Rechte
  Rechte beschreiben die Zugriffskontrolle auf Objekte, Operationen oder Attribute und werden mittels Operationen vergeben. Ist ein Nutzer im Besitz von Rechten, ist für diesen klar ersichtlich, welche Zugriffe erlaubt sind. Eine Unterteilung von Rechten liefert die - den POSIX-Rechten entsprechenden - Lese-, Schreib- und Ausführrechte.

- Objekte
  Objekte stellen Operationen und Attribute zur Verfügung. Hierbei ist eine Unterscheidung der möglichen Objekte äußerst wichtig, da verschiedene Objekte verschiedene Operationen und Attribute zur Verfügung stellen.

- Operationen
  Eine von einem Objekt zur Verfügung gestellte Operation ermöglicht die - dem Einsatzzweck entsprechende - Nutzung des Objekts.

- Attribute
  Attribute dienen der Beschreibung des Zustands sowie der Eigenschaften eines Objekts.

Welche denkbare Infrastruktur auch vorliegt, jede von ihnen lässt sich in die gelisteten Bestandteile zerlegen und zuordnen. Basierend auf diesen Bestandteilen entstand unter Berücksichtigung der im RNP verfügbaren Objekte, Operationen und Attribute eine Anwendungsprogrammierschnittstelle, welche VIFS zum Steuern beliebig vieler RNP Umgebungen befähigt und befindet sich im Anhang 1. Dabei wurde versucht generisch vorzugehen, um eine Schnittstelle zu erzeugen, welche in weiterer Folge für beliebige Infrastrukturen verwendet und gegebenenfalls erweitert werden kann.

Als abstraktes Basiselement wurde der in der Softwareentwicklung gänginge Typ “object” verwendet. Dabei fordert dieser Typ bereits die in jedem Objekt zur Verfügung stehenden Operationen “read_attribute” und “modify_attribute”. Aus diesem Basiselement wurden weitere Elemente wie bspw. “physical_machine” oder “virtualized_object” abgeleitet und mit den benötigten Operationen versehen. Hierbei wurde nochmals ersichtlich, wie unterschiedlich verschiedene Komponenten ausgeprägt sein können. Ein virtualisiertes Objekt (“virtualized_object”) stellt Operationen zur Verfügung, die bei einem physischen Rechner (“physical_machine”) nicht existieren wie die “Pausierung” der Maschine. Durch dieses heterogene Umfeld wird die Erzeugung eines geeigneten Rechtemodells zusätzlich erschwert. In einer teilweise virtualisierten Umgebung müssen beispielsweise domänenübergreifende Migrationen ermöglicht und die zugehörigen Befugnisse klar geregelt sein.

Als Ergebnis dieser Untersuchung geht die Domäne als das zentrale und alles umfassende Objekt hervor, welchem die interne Nutzer- und Rechteverwaltung unterliegt und welchem alle existierenden Objekte - dementsprechend auch andere Domänen - zugeteilt werden können. Die durch Domänen zur Verfügung gestellten Operationen sind hierbei auf feingranulare Delegationsmöglichkeiten und rollenbasierte Rechteverwaltung ausgelegt, um eine effiziente Verwaltung zu ermöglichen.

4.2 Anforderungen

Um das Ziel dieser Arbeit zu erreichen, müssen alle Anforderungen, welche durch die vorausgegangenen Kapiteln entstanden sind, nun klar formuliert werden. Aufgrund der entstandenen Schnittstelle und des Szenarios lassen sich die folgenden Anforderungen definieren.

4.2.1 Zugriffskontrolle

Eine vorhandene Rechtestruktur muss jedem Nutzer seine Zugriffsrechte auf infragekommende Objekte ersichtlich machen und diese Rechte durchsetzen. Es dürfen keine Zugriffe ermöglicht werden, die nicht explizit in der Rechtestruktur verankert sind. Desweiteren muss Administratoren ein Überblick über die bestehende Rechtestruktur ermöglicht werden, um eventuelle Fehlkonfigurationen in ihrer Domäne erkennen und beheben zu können.

4.2.2 Delegation

Da ein starres Rechtesystem der Idee von VIFS widerspricht, muss das Rechtemodell Modifikationen der bestehenden Rechtestruktur zulassen. Hierzu werden folgende Delegationsoperationen benötigt:

- Berechtigungsvergabe
  Um einem zuvor unbefugten Nutzer Zugriff auf Operationen zu gewähren, muss eine weitere Operation existieren, die es einem berechtigten Nutzer erlaubt, Lese-, Schreib- oder Ausführberechtigungen an andere Nutzer zu vergeben. Hierbei ist zusätzlich zu unterscheiden, ob die Berechtigungen vom empfangenden Nutzer nur verwendet oder gegebenenfalls verwendet und weitergegeben werden dürfen.

- Berechtigungen entziehen
  Um temporäre Befugnisse zurücknehmen zu können, muss es Wege geben, um einem Nutzer vergebene Rechte zu entziehen. Hierbei ist speziell darauf zu achten, wer welchem Nutzer welche Operationen entziehen darf.

- Besitzübergabe
  Einen Verantwortungsbereich zur Gänze einem anderen Nutzer zu übergeben, ermöglicht überschaubare Verantwortungsbereiche und ist somit eine unerlässliche Funktionalität. Hierbei sei erwähnt, dass diese Operation hauptsächlich von Verwaltungsdomänen zur Bildung neuer Verwaltungsdomänen benötigt wird. Die verbleibenden Zugriffsrechte von übergeordneten Domänen auf betroffene Objekte müssen situationsabhängig, anhand von einwirkenden Umständen, beispielsweise Datenschutzbestimmungen, festgelegt werden.

4.2.3 Domänenunterstützung

Um das volle Potenzial von VIFS auszuschöpfen, sollen mehrere - unter Umständen strikt getrennte - Domänen unterstützt werden. Dazu muss als Erstes sichergestellt werden, dass keine unerwünschten Befugnisüberschreitungen zwischen Domänen stattfinden können. Domänenadministratoren müssen dennoch in der Lage sein, Berechtigungen an domänenfremde Nutzer vergeben zu können, da die Zweckmäßigkeit von VIFS ansonsten erheblich an Wert verliert. Des Weiteren ist es zielführend, die Nutzerverwaltung domänenbasiert und autark zu arrangieren.

4.2.4 Rollentauglichkeit

Um das Rechtemanagement für die verantwortlichen Administratoren überschaubar und den Managementaufwand so gering wie möglich zu halten, sollten Rollenkonstrukte unterstützt werden. Zusätzlich sollten die Rollen frei definierbar und somit an alle Gegebenheiten anpassbar sein.

4.2.5 Dateisystem-Analogie

Da VIFS in ein Dateisystem abbildet, ist es erstrebenswert das Verhalten des darunterliegenden Berechtigungssystems nahe an der zugrundeliegenden POSIX-Zugriffskontrolle zu halten oder eventuell zur Gänze in diese zu integrieren, um so nicht nur die Struktur sondern auch

das Verhalten eines POSIX-Dateisystems beizubehalten. Durch diese Anforderung wird eine betriebssystemunabhängige Einsatzmöglichkeit und eine einfache Handhabung erzielt.

Werden alle aufgelisteten Anforderungen erfüllt, kann der Verwaltungsaufwand einer beliebigen Infrastruktur drastisch reduziert werden. Dies wird hauptsächlich durch feingranulare Delegationsoperationen ermöglicht, da somit autarke Domänen gebildet werden können, welche ihre internen Berechtigungen wiederum selbst verwalten können und so für den übergeordneten Domänenadministrator keinen wesentlichen Aufwand erzeugen. Rollenunterstützung bietet eine elegante Möglichkeit, um Nutzer ihren Aufgabenbereichen zuzuteilen und diese gegebenenfalls zu ändern. Aufgrund des Festhaltens an der Dateisystem-Metapher und dem Einhalten der POSIX-Kompatibilität, ist ein plattformunabhängiger Zugriff auf das zu erzeugende Dateisystem gesichert. Die Erfüllung aller Anforderungen ist natürlich wünschenswert, jedoch können durch die Einschränkungen seitens POSIX und FUSE bereits Probleme vorhergesehen werden.

5 Lösungsansätze

An dieser Stelle werden zwei verschiedene Lösungsansätze genauer untersucht. Kapitel 5.1 widmet sich einem theoretischen Ansatz und bezieht die gesamte Nutzer-, Gruppen-, und Rechteverwaltung in das von VIFS erzeugte Dateisystem mit ein, um so eine feingranulare Delegation zu ermöglichen. Aus Gründen der Umsetzbarkeit dieses Modells, geht Kapitel 5.2 im Gegensatz zu Kapitel 5.1 auf die von POSIX zur Verfügung gestellten Möglichkeiten ein um die Anforderungen realisierbar zu erfüllen.

5.1 Theoretisches Modell

Dieses Kapitel widmet sich dem theoretischen Modell, welches allen entstandenen Anforderungen entspricht. Dahingehend werden die - jeder Infrastruktur inhärenten - Elemente Nutzer, Rechte, Objekte, Operationen und Attribute sowie die zusätzlich aus den Anforderungen enstandenen Domänen und Rollen berücksichtigt.

5.1.1 Grundgedanke

Da es sich bei VIFS um ein Managementtool handelt, welches in der Lage sein soll, eine unbegrenzte Anzahl von heterogenen Infrastrukturen - welche wiederum eine Vielzahl von Domänen beherbergen können - zu verwalten, bieten sich modifizierte Access-Control-Lists (kurz ACL, siehe [acl03]) für den Einsatz als Zugriffskontrolle an. Hierbei soll eine generische Struktur, welche sich auf jede beliebige in VIFS abgebildete Infrastruktur anwenden lässt, entstehen. Zieht man die in Kapitel 4 entstandene API zur Erstellung dieser Struktur heran, kristallisieren sich Domänen als die alles beinhaltenden Objekte heraus. Diese beinhalten sowohl Nutzer als auch Komponenten und gegebenenfalls weitere Subdomänen. Dementsprechend sollte eine Domäne die - in ihr geltende - Rechtestruktur sowohl beinhalten als auch verwalten können. Abbildung 5.1 zeigt den Entwurf eines maßgeschneiderten Modells. Dabei sind die mit Tilde versehenen Bezeichnungen als beliebig benennbar zu interpretieren und es ist von einem Mehrfachvorkommen dieser Elemente auszugehen. In diesem Modell werden, genau wie Komponenten und Operationen, auch Rechte, Nutzer und Rollen auf Verzeichnisstrukturen, Dateien und Verlinkungen abgebildet. Hierbei enthält das Verzeichnis eines Nutzers seine Befugnisse innerhalb der gesamten (auch domänenübergreifenden) Struktur. Eine Unterscheidung zwischen “ausführbaren” und “delegier- und ausführbaren” Operationen ermöglicht eine sehr feingranulare Vergabe von Rechten. Ein Nutzer kann so befähigt werden, gezielte Operationen auszuführen und weiterzugeben und andere Operationen nur auszuführen. Das selbe gilt für Rollenkonstrukte. Die Zugehörigkeit eines Nutzers zu einer Rolle kann im zugehörigen “Roles”-Verzeichnis des Nutzers ausgelesen werden. Zusätzlich besitzt jedes Element dieser Struktur die durch die POSIX-Zugriffskontrolle zur Verfügung gestellten Elemente Eigentümer und Gruppe sowie die zugehörigen Rechte. Beide Merkmale zeigen hierbei auf die Betreiber der Domäne und erlauben diesen uneingeschränkten Zugriff auf alle zugehörigen Elemente. Durch diese Strukturierung werden Verwaltungsdomänen auf

die Eigentümer bzw. Gruppenmitglieder der POSIX-Dateisystemrechte abgebildet. Für die Abbildung der Organisationsdomänen lassen sich die erzeugten Rollenkonstrukte sowie gegebenenfalls sogar Subdomänen verwenden.

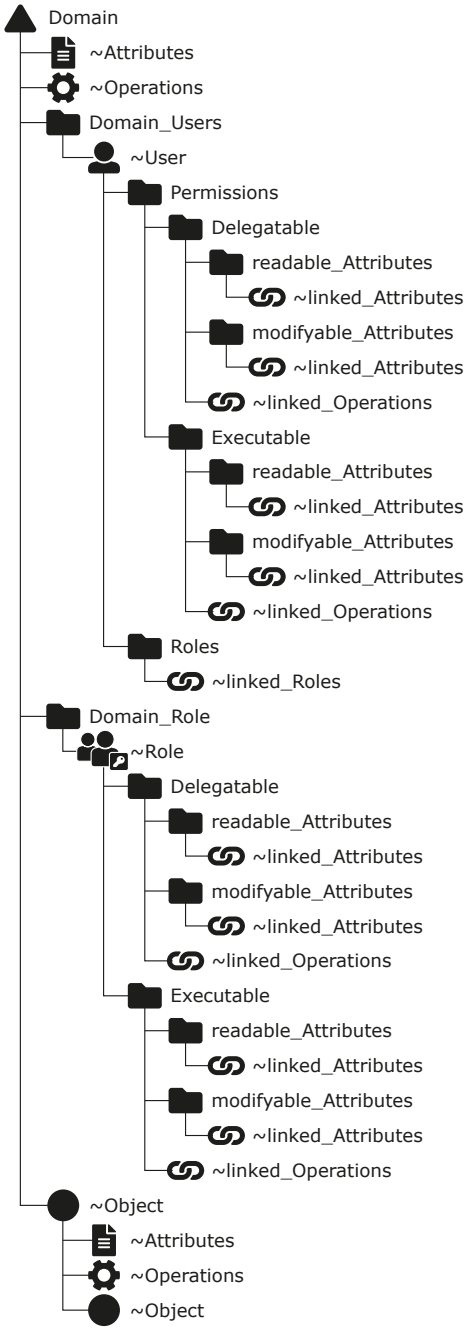

Abbildung 5.1: ACL-Struktur

Die in Abbildung 5.1 bewusste Trennung von “delegatable” und “executable” betrifft die Verwendungsmöglichkeit von Zugriffsberechtigungen. Besitzt ein Benutzer eine Verlinkung auf eine Operation im “delegatable”-Verzeichnis, kann er sie ausführen und unter den richtigen Voraussetzungen auch weitergeben. Die Art der Weitergabe (delegierbar oder nur ausführbar) entscheidet sich durch die Verwendung der entsprechenden Operationen:

- “grant_permission”:
  Wird diese Operation zur Erteilung von Zugriffsrechten verwendet, wird die entsprechende Verlinkung im Verzeichnis “executable” erstellt und der Empfänger verfügt lediglich über das Recht zur Ausführung der entsprechenden Zieloperation. Hierbei spielt es keine Rolle, ob der empfangende Nutzer zur Ausführung der Operationen “grant_permission” oder “grant_permission_delegatable“ befugt ist.
  Eine durch ”grant_permission“ delegierte Operation kann vom Empfänger stets nur ausgeführt werden.

- “grant_permission_delegatable”:
  Wird diese Funktion verwendet, darf der Empfänger - vorausgesetzt er verfügt über das Recht zur Ausführung von “grant_permission” oder “grant_permission_delegatable” - dieses Recht beanspruchen (Zieloperation ausführen) sowie weiterreichen.

Beide Operationen sind auf Nutzer sowie auf Rollen anwendbar. Die Vollständigkeit dieser Delegationssemantik erschließt sich aus der in Kapitel 5.1.2 befindlichen Delegationsalgebra. Die Unterteilung von lesbaren und modifizierbaren Attributen dient der zusätzlichen Sicherheit des Systems. Beispielsweise könnte das Auslesen der IP-Adressen bestimmter Domänenbestandteile Informationen über den strukturellen Aufbau der originalen Infrastruktur preisgeben, was unter Umständen nicht erwünscht ist. Um der Dateisystem-Metapher zu entsprechen, müssten Dateimanager-Programme sowie Konsolen beim Navigieren durch die Hierarchie so manipuliert werden, dass sie stets die für den aktuellen Nutzer gültigen Rechte anzeigen. Dies könnte mittels der Rechtegruppe “Andere” realisiert werden. Domänen-Administratoren müsste der Überblick über vergebene Rechte mittels einer Auflistung aller Verlinkungen auf domäneninterne Operationen gewährt werden.

5.1.2 Delegationsalgebra

Die Tabelle 5.1 zeigt die zwei unterschiedlichen Delegationsoperationen, angewandt auf delegierbare und nicht-delegierbare Operationen.

Ist ein Benutzer im Besitz der Rechte in der Spalte “Rechte”, ist er befähigt die Operationen in der Spalte “Ausführung” auszuführen. Die Spalte “Ausführung” zeigt also alle Operationen, die aufgrund der vorhandenen Rechte in Spalte “Rechte” ausgeführt werden können. Das Ergebnis der Ausführung ist in der Spalte “Rechte(Zielnutzer)” zu sehen und beschreibt die sich aus der Operation ergebenden Rechte für den Zielnutzer. Die ausführbaren Operationen des Zielnutzers können somit wieder aus der Spalte “Rechte” abgelesen werden.

Mengendefinitionen:
Sei O die Menge der in der Struktur verfügbaren Operationen.
O^{+}sei Teilmenge von O und enthält die delegier- und ausführbaren Operationen o^{+}.
O^{-} sei Teilmenge von O und enthält die ausführbaren Operationen o^{-}.
O^{*} sei Teilmenge von O, welche Operationen enthält, die nur auf delegier- und ausführbare (o^{+}) Operationen angewandt werden können und wiederum in O^{*+} und O^{*-} unterteilt werden kann.

o sei eine beliebige aber feste Operation der Menge $O \setminus O^{*}$
gp (grant permission): $gp \in O^{*}$
gdp (grant delegation permission): $gdp \in O^{*}$

Schlussfolgerung:
$o^{-} \in O^{-} \setminus O^{*}, o^{+} \in O^{+} \setminus O^{*}, gp^{-} \in O^{*-} \Rightarrow gq^{-} \in O^{-}$, $gp^{+} \in O^{*+} \Rightarrow gp^{+} \in O^{+}$,
$gdp^{-} \in O^{*-} \Rightarrow gdp^{-} \in O^{-}, gdp^{+} \in O^{*+} \Rightarrow gdp^{+} \in O^{+}$
($\Rightarrow$ Delegationsoperationen sind ebenfalls delegierbar)

Aufgrund der Übersichtlichkeit wird der Zielnutzerparameter bei den folgenden Funktionsdefinitionen nicht aufgeführt.

Funktionsdefinitionen:

$o^{+/-}$:	Operation o wird ausgeführt
$gp^{+/-}(x^{+}) \rightarrow x^{-}$:	Die Operation gp nimmt delegier- und ausführbare Operationen und gibt ausführbare Operationen zurück
$gdp^{+/-}(x^{+}) \rightarrow x^{+}$:	Die Operation gdp nimmt delegier- und ausführbare Operationen und gibt delegier- und ausführbare Operationen zurück

Tabelle 5.1: Delegationsfunktionen und Auswirkungen

Rechte	Ausführung	=	Rechte(Zielnutzer)
o^+	o^+	-	
o^-	o^-	-	
gp^-		-	
gp^+	$gp^+(gp^+)$	E	gp^-
gdp^-		-	
gdp^+	$gdp^+(gdp^+)$	D	gdp^+
gp^- , o^-	o^-	-	
gp^- , o^+	o^+	-	
	$gp^-(o^+)$	E	o^-
gp^+ , o^-	o^-	-	
	$gp^+(gp^+)$	E	gp^-
gp^+ , o^+	o^+	-	
	$gp^+(o^+)$	E	o^-
	$gp^+(gp^+)$	E	gp^-
gdp^- , o^+	o^+	-	
	$gdp^-(o^+)$	D	o^+
gdp^- , o^-	o^-	-	
gdp^+ , o^-	o^-	-	
	$gdp^+(gdp^+)$	D	gdp^+
gdp^+ , o^+	o^+	-	
	$gdp^+(o^+)$	D	o^+
	$gdp^+(gdp^+)$	D	gdp^+
gp^- , gdp^- , o^-	o^-	-	
gp^- , gdp^- , o^+	o^+	-	
	$gp^-(o^+)$	E	o^-
	$gdp^-(o^+)$	D	o^+
gp^- , gdp^+ , o^-	o^-	-	
	$gp^-(gdp^+)$	E	gdp^-
	$gdp^+(gdp^+)$	D	gdp^+
gp^- , gdp^+ , o^+	o^+	-	
	$gp^-(o^+)$	E	o^-
	$gp^-(gdp^+)$	E	gdp^-
	$gdp^+(o^+)$	D	o^+
	$gdp^+(gdp^+)$	D	gdp^+
gp^+ , gdp^- , o^-	o^-	-	
	$gdp^-(gp^+)$	D	gp^+
	$gp^+(gp^+)$	E	gp^-
gp^+ , gdp^- , o^+	o^+	-	
	$gp^+(o^+)$	E	o^-
	$gp^+(gp^+)$	E	gp^-
	$gdp^-(o^+)$	D	o^+
	$gdp^-(gp^+)$	D	gp^+
gp^+ , gdp^+ , o^-	o^-	-	
	$gp^+(gp^+)$	E	gp^-
	$gp^+(gdp^+)$	E	gdp^-
	$gdp^+(gp^+)$	D	gp^+
	$gdp^+(gdp^+)$	D	gdp^+
gp^+ , gdp^+ , o^+	o^+	-	
	$gp^+(o^+)$	E	o^-
	$gp^+(gp^+)$	E	gp^-
	$gp^+(gdp^+)$	E	gdp^-
	$gdp^+(o^+)$	D	o^+
	$gdp^+(gp^+)$	D	gp^+
	$gdp^+(gdp^+)$	D	gdp^+

5.1.3 Umsetzbarkeit

Die Realisierung dieses Modells gestaltet sich in der momentan Situation als äußerst schwierig. Aufgrund der aktuell fehlenden ACL-Unterstützung der verwendeten FUSE-Variante, müsste die POSIX-Zugriffskontrolle für die Zugriffskontrolle aller Organisationsdomänen umgangen werden, was sich als äußerst sicherheitskritisch erweist. Da dieses Modell, abgesehen von der Domänenintegration, den in verschiedenen UNIX-Derivaten verwendeten ACL-Strukturen sehr ähnelt, würde sich die Komplexität der Implementierung bei ACL-Unterstützung deutlich verringern. So könnten Berechtigungen und Rollen durch einen Hintergrundmechanismus aus den Verzeichnissen und Verlinkungen ausgelesen und in entsprechende ACL übersetzt werden. Hierfür müsste jedoch zuerst sichergestellt werden, dass diese Verlinkungen nur von Nutzern erstellt werden können, die die benötigten Rechte auf entsprechende Operationen oder Attribute, oder selbst eine Verlinkung im zugehörigen "delegatable"-Verzeichnis auf die zu delegierenden Elemente und Zugriffsrechte auf eine der beiden genannten Delegationsoperationen der entsprechenden Domäne besitzen. Je nach Größe der abgebildeten Infrastruktur kann eine solche Validierung nicht zu vernachlässigende Rechenzeit beanspruchen. Da POSIX zusätzlich keine offiziellen Richtlinien zur Integration von ACL vorgibt, würde man durch dieses Vorgehen Gefahr laufen, die von VIFS angestrebte Plattformunabhängigkeit zu verlieren. Aus diesem Grund dient dieses Modell zum jetzigen Zeitpunkt als Veranschaulichung der vorliegenden Komplexität sowie als Vergleichsmöglichkeit bei der Erfüllung der Anforderungen.

5.2 Abbildung in POSIX

Da das vorausgegangene und allen Anforderungen entsprechende Modell aktuell nicht realisierbar ist, widmet sich dieses Kapitel der möglichen Abbildung in die “traditionelle POSIX Zugriffskontrolle”.

5.2.1 Grundgedanke

Es wird erneut auf das OSI-Management-Schema zurückgegriffen, um die zwei Arten von Domänen abzubilden. Die Verwaltungsdomänen müssen in der Lage sein, die benötigten Organisationsdomänen zu erzeugen und diese den entsprechenden Dateisystemelementen zuzuteilen. Da lediglich der Dateieigentümer in der Lage ist, mittels Änderung der zugehörigen Gruppe zu delegieren, existiert für die Abbildungen dieser Domänen keine andere Möglichkeit. Gegebenenfalls muss hierbei ein Nutzerkonto von mehreren Personen benutzt werden. Für die Abbildung der Organisationsdomänen verbleiben somit nur die POSIX-Gruppen. Hierbei wird jedoch selten eine injektive Abbildung möglich sein, da es relativ häufig vorkommt, dass sich Organisationsdomänen überschneiden. An dieser Stelle schränkt POSIX sehr stark ein. Eine mögliche Lösung ist die Erzeugung neuer Gruppen, welche alle benötigten Nutzer vereinen und den betroffenen Dateisystemelementen zugewiesen werden. Die Alternative wäre von Beginn an eine Gruppe pro Dateisystemelement zu vergeben und diese zu befüllen. Die Abbildung von verschiedenen Rollen innerhalb der Organisationsdomänen ist somit nicht realisierbar. Dementsprechend müssen Organisationsdomänen so gewählt werden, dass sie den gewünschten Rollen entsprechen. Ein detaillierter Vergleich mit den gegebenen Anforderungen schafft hierbei zusätzliche Übersicht über die vorhandene Problematik.

5.2.2 Anforderungsvergleich

- Zugriffskontrolle:
  Die Zugriffskontrolle wird grundlegend von POSIX selbst erledigt. Ausnahmen könnten hier jedoch nötig werden, da beispielsweise die Besitzübergabe Rechte benötigt, welche durch POSIX nicht auf Domänen beschränkt werden können. Die Nutzer- und Gruppenverwaltung kann jedoch ausgelagert werden, sodass bei der Erzeugung von Gruppen und dem Zuweisen von Nutzern diese Rechte nicht benötigt werden.

- Delegation:
  Zur Delegation bietet POSIX mehrere Möglichkeiten. Einerseits können die Rechte einer Datei verändert werden und so den Zugriff erlauben. Andererseits kann mittels Änderung an der zugehörigen Gruppe - also durch Hinzufügen von Nutzern sowie Wechseln der Gruppe - oder der Übergabe des Objekts an einen anderen Eigentümer eine delegierende Tätigkeit umgesetzt werden. Eine Besitzübergabe kann hierbei problematisch werden, da für diese Operation schwer einschränkbare Rechte benötigt werden. Somit muss für diesen Punkt eine geeignete Lösung (VIFS-übergreifender Administrator oder entsprechende Implementierung) gefunden werden. Um Berechtigungen zu vergeben oder zurückzuziehen, können die POSIX-Gruppen benutzt werden. Hierbei werden keine erweiterten Rechte benötigt, da der Eigentümer des Objekts die zugehörige Gruppe ändern darf, sofern er selbst Mitglied der neuen “Zielgruppe” ist.

- Domänenunterstützung:
  Aufgrund der strikten Zugriffskontrolle von POSIX ist Domänenunterstützung durchaus realisierbar. Um die Übersicht und Wartbarkeit des Systems jedoch auch bei größeren Infrastrukturabbildungen zu gewährleisten, ist es ratsam, die Gruppen- und Nutzerverwaltung an ein dafür spezialisiertes System auszulagern. Stellt dieses System ACL zur Verfügung, ist eine annähernd autarke Domänenverwaltung möglich.

- Rollentauglichkeit:
  Durch die bereits erlangten Erkenntnisse ist erkennbar, dass eine feingranulare Rollenbildungsoption zur Unterteilung von Organisationsdomänen nicht möglich ist. Eine Rolle entspricht in diesem Modell also exakt einer Organisationsdomäne. Für eine temporäre Zugriffserteilung auf ein Element an einen Nutzer, muss eine neue Gruppe erzeugt werden, der sowohl die ursprünglich berechtigten Gruppennutzer, wie auch der/die neue/n Nutzer hinzugefügt werden müssen. Dadurch wird die ursprüngliche "eins zu eins"-Darstellung von Rolle/Organisationsdomäne zu Gruppe zerstört und die Übersichtlichkeit für den Administrator leidet stark darunter.

- Dateisystem-Analogie:
  Die Dateisystem-Metapher bleibt erhalten, insofern Nutzer nicht gezwungenermaßen über externe Systeme ihre Gruppenzugehörigkeit und somit ihre Befugnisse eruiren müssen. Ist ein Benutzer Mitglied vieler verschiedener Gruppen, kann hierbei unter Umständen nicht auf die betriebssystemeigene Gruppenverwaltung zum Prüfen der Zugehörigkeit zugegriffen werden. Je nach Implementierung können dem Nutzer jedoch mittels entsprechendem Befehl alle Gruppen denen er angehört oder alle Dateisystemelemente auf die er Zugriff hat angezeigt werden.

  Die Abbildung einer beliebigen Infrastruktur und deren Rechtestruktur in das POSIX-Rechtemodell ist also mit Einschränkungen realisierbar. Die Einschränkungen sowie der Verwaltungsaufwand variieren je nach Implementierung und Komplexität der abzubildenden Struktur. Grundsätzlich ist es vorteilhaft, die Verwaltung von Gruppen und Nutzern an dafür spezialisierte Systeme auszulagern, um mehr Kontrolle über die domäneneigenen Gruppenstrukturen zu ermöglichen.

  Ein exemplarischer Auszug der mathematischen Darstellung der Abbildung einer reduzierten RNP-Umgebung in VIFS befindet sich im folgenden Kapitel 5.2.3. Eine praktische Umsetzung folgt in - dem Prototyp gewidmeten - Kapitel 6.

5.2.3 Mathematische Abbildung

Aufgrund der in den Kapiteln 2.3 und 2.4 definierten Mengen, lässt sich die im Prototyp erzeugte Abbildung auch mathematisch darstellen. Um einen Rückblick in die entsprechenden Kapitel zu ersparen, werden hier die Mengendefinitionen nochmals kurz ausgeführt.

Mengendefinition: Domänen

$P :=$ $\{i \mid i \in N\}$ (Personen)

$K :=$ $\{(id, O_{\mathrm{k}}) \mid id \in N\}$ (Komponenten samt Operationen)

$D_{\mathrm{v}} =$ $\{(\{p\}, \{k_{+}{}^{\mathrm{i}}\}) \mid p \in P,\ k \in K\}$ (Verwaltungsdomänen)

$D_{\mathrm{o}} =$ $\{(\{p\}, \{k_{\text{-}}{}^{\mathrm{i}}\}) \mid p \in P,\ k \in K\}$ (Organisationsdomänen)

Mengendefinition: Dateisystem

$F_{\mathrm{i}} :=$ $\{f \mid f \in F\}$ (Dateisystemelemente)

$U_{\mathrm{i}} :=$ $\{UID \mid UID \in N\}$ (Dateisystemnutzer)

$G_{\mathrm{i}} :=$ $\{(GID, \{u\}) \mid GID \in N,\ u \in U_{\mathrm{i}}\}$ (Dateisystemgruppen)

$A :=$ $\{((abc), (def), (ghi)) \mid$
$a, d, g \in \{r, -\}, b, e, f \in \{w, -\}, c, f \in \{x, s, S, -\}, i \in \{x, t, T, -\}\}$
(Dateisystemrechte)

$V := \{(\{(f, u, g, a)\}) \mid f \in F_{\mathrm{i}}, u \in U_{\mathrm{i}}, g \in G_{\mathrm{i}}, a \in A\}$
(Dateisystem samt Nutzer, Gruppen und Rechten)

Abbildungsfunktionen

$x : D_{\mathrm{v}} \to V : d_{\mathrm{v}}{}^{\mathrm{i}} \to x(d_{\mathrm{v}}{}^{\mathrm{i}})$
$y : D_{\mathrm{o}} \to V : d_{\mathrm{o}}{}^{\mathrm{i}} \to x(d_{\mathrm{o}}{}^{\mathrm{i}})$

Funktion x bildet Verwaltungsdomänen auf die Besitzer und Funktion y bildet die Organisationsdomänen auf Gruppen der Dateisystemelemente ab.

Ausgangsmengen

Da in Kapitel 3 das RNP bereits ausführlich beschrieben und dabei festgestellt wurde, dass das Praktikum über eine einzelne Verwaltungsdomäne verfügt, wird mit der Darstellung der Verwaltungsdomänen begonnen. Auch das MNM soll in diesem Fall mit einer Verwaltungsdomäne auskommen.

a) Verwaltungsdomänen

Wenn D_v die Menge aller in der abzubildenden Struktur vorhandenen Verwaltungsdomänen ist, so besteht sie in diesem Fall aus zwei Elementen:

- MNM-Verwaltungsdomäne
- RNP-Verwaltungsdomäne

Zusätzlich wird davon ausgegangen, dass das MNM in diesem Fall keine steuerbaren Komponenten verwaltet sondern lediglich Personal und Räumlichkeiten zur Verfügung stellt. Dennoch wird diese Domäne explizit erzeugt, da das Hinzufügen von MNM-verwalteten Komponenten nicht auszuschließen ist und es die dem RNP übergeordnete Instanz ist. Sollten also Personalausfälle im RNP stattfinden, liegt es an der MNM-Verwaltungsdomäne entsprechende Maßnahmen zu ergreifen.

$$D_\mathrm{v} = \{(\{mnmadmin01, mnmadmin02\}, \{{k_+}^{\text{barraca*}}, {k_+}^{\text{virt-lab*}}\}),\\ (\{rnpadmin01, rnpbetreuer01\}, \{{k_+}^{\text{RNP*}}, ...\})\}$$

Der Stern im Komponentenanteil der Verwaltungsdomänenelemente repräsentiert hierbei alle zugehörigen abzubildenden Objekte und Attribute.

b) Organisationsdomänen

Die benötigten Organisationsdomänen können hierbei durch die verschiedenen Gruppierungen der Studenten sowie die Gruppierung der Tutoren abgebildet werden. Da der RNP-Betreuer bereits in die Verwaltungsdomäne aufgenommen wurde und auf keinen zusätzlichen Komponenten, welche nicht bereits durch die Verwaltungsdomäne abgedeckt werden, agieren muss, ist es unnötig an dieser Stelle zusätzliche Organisationsdomänen zu erzeugen. Es werden also folgende Organisationsdomänen erzeugt:

- RNP-Tutoren
- RNP-Studenten-Gruppe01
- RNP-Studenten-Gruppe02

$$D_\mathrm{o} = \{(\{rnptutor01, rnptutor02\}, \{{k_-}^{\text{Gruppe01}}, {k_-}^{\text{Gruppe02}}\}),\\ (\{std01, std02\}, \{{k_-}^{\text{Gruppe01-router01}}, ..., {k_-}^{\text{Gruppe01-pc06}}\}),\\ (\{std03, std04\}, \{{k_-}^{\text{Gruppe02-router01}}, ..., {k_-}^{\text{Gruppe02-pc06}}\})\}$$

Abbildung

Die Mengennotation des POSIX-Dateisystems wurde in Kapitel 2 erläutert und dient nun als Zielmenge. Die in VIFS vorhandenen Dateisystemelemente können Kapitel 6 entnommen werden. Die Abbildung findet in zwei Schritten statt, wobei als erstes die Verwaltungsdomänen und im zweiten Schritt die Organisationsdomänen abgebildet werden.

a) Abbildung der Verwaltungsdomänen

Die Mitglieder der Verwaltungsdomänen müssen sich zur Umsetzung ihrer Aufgaben einen Nutzer teilen. Dementsprechend werden mehrere Personen auf einen einzelnen Nutzer abgebildet.

$x((\{mnmadmin01, mnmadmin02\}, \{{k_+}^{\text{MNM-Domäne}}, {k_+}^{\text{Barraca}}, {k_+}^{\text{virt-lab}}\}) =$
$\{(../\text{MNM}/, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi))),$
$(../\text{MNM}/\_\text{contact}, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi))),$
$(../\text{MNM}/\_\text{location}, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi))),$
$...,$
$(../\text{MNM/barraca}/, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi))),$
$(../\text{MNM/barraca}/\_\text{contact}, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi))),$
$...,$
$(../\text{MNM/barraca/virt-lab}/\_\text{contact}, \text{mnmvduser}, g, ((\text{rwx}), (def), (ghi)))\}$

$x((\{rnpadmin01, rnpbetreuer01\}, \{{k_+}^{\text{RNP}}, ..., {k_+}^{\text{RNP-Gruppe01}}...\})) =$
$\{(../\text{MNM/barraca/virt-lab/RNP}/*, \text{rnpvduser}, g, ((\text{rwx}), (def), (ghi)))\}$

Bei der Abbildung der RNP-Verwaltungsdomäne symbolisiert der Stern alle in dem Verzeichnis RNP enthaltenen Dateisystemelemente (rekursiv).

b) Abbildung der Organisationsdomänen

Die Mitglieder der Organisationsdomänen werden auf die Gruppen der Dateisystemelemente abgebildet. Die Vergabe der Zugriffsrechte kann im Nachhinein von der Verwaltungsdomäne verändert werden. Zu diesem Zeitpunkt wird von Lese- und Ausführzugriff ausgegangen. Die Zerlegung der Elemente k_ findet nur andeutungsweise statt. Da in VIFS jede von Organisationsdomänen benötigte Operation einer Datei entspricht, werden hierbei die Subelemente von ${k\_}^{\text{i}} = (id, \{o^{\text{j}}, ...\}))$, also o^{j} auf die Pfade der äquivalenten Operationen abgebildet.

$y((\{rnptutor01, rnptutor02\}, \{{k\_}^{\text{Gruppe01}}, {k\_}^{\text{Gruppe02}}\})) =$
$\{(../\text{RNP/Gruppe01/+REBOOT}, u, \text{tutoren}, ((abc), (\text{r-x}), (ghi))),$
$(../\text{RNP/Gruppe01/vs12/+DOWN}, u, \text{tutoren}, ((abc), (\text{r-x}), (ghi))),$
$...\}$

$$y((\{std01,\ std02\},\ \{k\_^{\text{Gruppe01-router01}},\ ...,\ k\_^{\text{Gruppe01-pc06}}\}) =$$
$$\{(../\text{RNP/Gruppe01/router01/+REBOOT},\ u, \text{grp01},\ ((abc),\ \text{(r-x)},\ (ghi))),$$
$$...,$$
$$(../\text{RNP/Gruppe01/pc06/vif9.0/+DOWN}, u, \text{grp01}, ((abc), \text{(r-x)}, (ghi))),$$
$$...\}$$

6 Prototyp

Aufgrund der gewonnenen Erkenntnisse wird die Implementierung mittels einer Trennung von Zugriffskontrolle und Gruppen- sowie Nutzerverwaltung durchgeführt. Die aufgezeigte Problematik bei der Besitzübergabe von Dateisystemelementen wird dabei durch eine Anfrage an ein Mitglied der VIFS-Administrationsgruppe realisiert. Eine entsprechend sichere Implementierung, welche diesen Schritt ersparen würde, bleibt zukünftigen Arbeiten vorbehalten. Der Betreiber der zu erzeugenden VIFS-Struktur sei das MNM-Team, das Einbinden von Infrastrukturen aus anderen Lehr- und Forschungsabteilungen soll aber ermöglicht werden.

Als Träger des hierbei entstehenden Prototyps dient ein RaspberryPi Modell 3 mit 1024 MB RAM und einem ARM7 Prozessor. VIFS wurde modifiziert, um eine persistente Speicherung des erzeugten Rechtekonstrukts zu erlauben. Diese Implementierung erspart die wiederholte und aufwendige Erzeugung der Rechtestruktur nach einem Neustart des Hosts oder VIFS. Als Betriebssystem wird Raspbian verwendet (ein für RaspberryPi optimierter Debian-Ableger). Für die Umsetzung der Trennung von Zugriffskontrolle und Nutzermanagement wurde LDAP verwendet und aus den offiziellen Paketquellen (Debian) installiert. Weitere Details über Konfigurationen folgen in den entsprechenden Subkapiteln.

Es wird davon ausgegangen, dass VIFS beim Einsatz als Lehrhilfsmittel keine Limitierungen für das Betreten und Auslesen von Verzeichnissen umsetzen muss. Dementsprechend ist es jedem Benutzer erlaubt, in jedes von VIFS erzeugte Verzeichnis zu navigieren und sich Dateien bzw. Verzeichnisse sowie deren Rechte anzeigen zu lassen. Die Rechtegruppe “Andere” wird bei Verzeichnissen stets mit den Rechten “r-x” ausgestattet. Diese Basiskonfiguration kann natürlich entsprechend den Bedürfnissen des Betreibers verändert und beispielsweise durch eine ledigliche “Durchgangserlaubnis” oder eine komplette “Zutrittssperre” für unbefugte Nutzer ersetzt werden.

6.1 Modifikationen an VIFS

Um nach dem Neustart von VIFS oder des Trägergeräts die manuelle Neuvergabe von Rechten zu vermeiden, wurde eine persistente Speicherung der Rechte welche über das VIFS-Dateisystem gelegt werden, implementiert. Als Speicher wurden SQLite3-Datenbanken gewählt. Die Speicherung bei Vergabe von Rechten erfolgt zeitgleich durch VIFS und bleibt dem ausführenden Nutzer verborgen. Des Weiteren wurde ein Automatismus integriert, welcher bei Einbindung zusätzlicher Infrastrukturen den Eigentümer und die Gruppe des beinhaltenden Verzeichnisses auf neue Strukturen und Dateisystemelemente anwendet. Durch diesen Vorgang ist bei der Anfrage zur Einbindung einer neuen Umgebung an VIFS-Administratoren der Eigentümer der neuen

Umgebung bereits im Vorfeld definiert und ein weiterer Arbeitsschritt bleibt unter Umständen erspart. Die zugehörigen mit Kommentar versehenen Codefragmente befinden sich in Anhang 2.

6.2 LDAP-Basiskonfiguration

Zur Verwaltung der Nutzer und Gruppen wurden die Standardpakete aus den Debian Repositories “slapd” und “ldap-utils” verwendet. Konfiguriert wurde slapd für die Domäne “vifs”. Somit endet jeder Authentifizierungsstring mit dem Fragment “dc=vifs”. Weitere Domänenunterteilungen erfolgten im laufenden Betrieb mittels der Frontend-Software “ldap-account-manager”, welche ebenfalls über die Standardrepositories installiert wurde. Damit der VIFS-Träger gegen LDAP authentifiziert, wurde der Hostname unter /etc/hostname der Domäne entsprechend umbenannt. Als Authentifizierungsprogramm wurde “libnss-ldapd” - ebenfalls aus den Standardrepositories von Debian - installiert und konfiguriert. Weitere Konfigurationsdetails - wie die Einrichtung des “LDAP Connection Daemon” und des “Name Service Switch” können unter den zugehörigen Dokumentationen nachgelesen werden und werden hier nicht weiter ausgeführt, da es sich um Standardkonfigurationen handelt.

Der nächste Schritt umfasst die Erstellung der ersten autarken Domäne von VIFS, die VIFS-Administrations-Domäne. Hierfür wurden weitere “organisational units” (im Folgenden “ou”) unter den bereits vorgegebenen ou Groups und Users mit dem Namen “administration” erstellt. Im User-Pfad wurde eine weitere ou für die Nutzer innerhalb der Administrationsdomäne erzeugt (siehe 6.1). Daraufhin wurden die VIFS-Administratoren “vifsadmin01” und “vifsadmin02” in der entsprechenden Nutzer-ou erzeugt und der Gruppe “vifsadmin” hinzugefügt. Des Weiteren wurde vorausschauend eine ou (jeweils in Users & Groups) für die Unterbringung von zukünftigen Domänen erzeugt und die MNM-Domäne wurde dieser ou hinzugefügt. So können weitere VIFS-Beitrittsanfragen von anderen Lehr- und Forschungsabteilungen sauber getrennt verwaltet werden.

Um ausreichende Berechtigungen für die Domäne der VIFS-Administratoren - aktuell also das gesamte Konstrukt - zu vergeben, wurden die in der Auflistung 6.1 gezeigten ACL in der von LDAP verwendeten Konfigurationsdatei ergänzt.

Listing 6.1: Basis-ACL

```
olcAccess: to attrs=userPassword,shadowLastChange
    by self write
    by dn.subtree="ou=users,ou=administration,ou=Users,dc=vifs"
        write
    by anonymous auth
    by * none
olcAccess: to *
    by dn.subtree="ou=users,ou=administration,ou=Users,dc=vifs"
        manage
    by dn.subtree="cn=vifsadmin,ou=administration,ou=Groups,dc=vifs"
        manage
    by * read
```

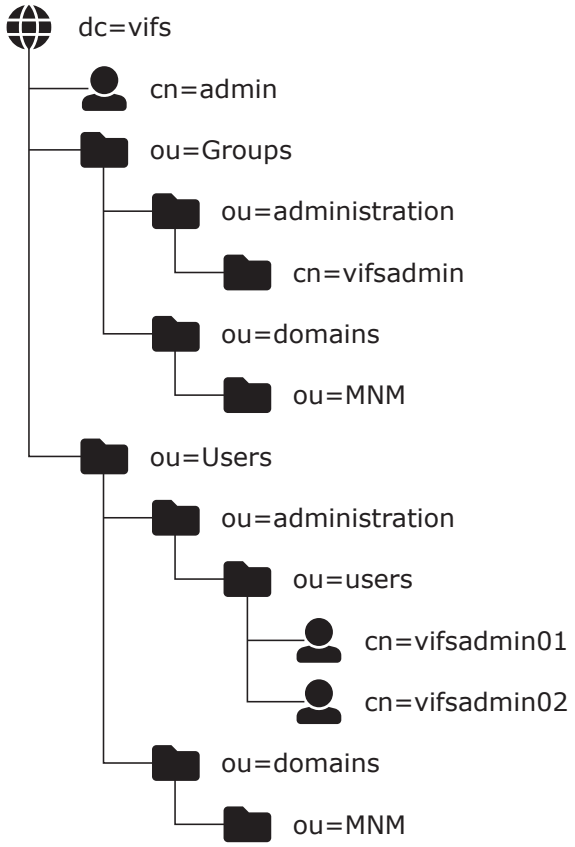

Abbildung 6.1: Basis-Struktur für VIFS-Administration

Des Weiteren wurde das Login auf der Frontend-Konfigurationsseite für LDAP-Benutzer freigegeben, um die Konfigurationen der VIFS-Administratoren über die vom Paket "ldap-account-manager" zur Verfügung gestellte Benutzeroberfläche zu erlauben. Im Anschluss wurde die von VIFS abgebildete Basisinfrastruktur (ohne Verbindung

zu RNP-Rechnern) eingehängt und der Inhalt an den Benutzer “vifsadmin01” und die Gruppe “vifsadmin” übergeben. In der “sudoers”-Datei wurde sichergestellt, dass Mitglieder der Gruppe “vifsadmin” über ausreichende Rechte verfügen, um eine Besitzübergabe von VIFS-Dateisystemelementen zu ermöglichen. In den folgenden Schritten wurden alle Operationen unter dem Benutzer “vifsadmin01” ausgeführt.

6.3 Erweiterung der VIFS-Umgebung

Um auf Anfrage des RNP-Administrators eine weitere Subdomäne zu erzeugen, wurde die VIFS-Konfigurationsdatei “org.conf” entsprechend konfiguriert (6.2). Die in dieser Konfiguration erwähnte “cluster.conf” beinhaltet die vom RNP-Administrator gewünschten anzubindenden Maschinen sowie deren strukturelle Gliederung. Für die Verbindung von VIFS zu den zu verwaltenden Maschinen musste die ssh-Konfigurationsdatei des Trägersystems angepasst werden (nicht angeführt).

Listing 6.2: Strukturkonfigurationsdatei von VIFS

```
{ "MNM":
   { "_attr" :
       { "_orgname" : "MNM-Team",
         "_type" : "datacentre",
         "_location" : "Oettingenstra e 67",
         "_contact" : "abuse@mnm-team.org"
       },
     "_sons" :
       { "barraca" :
           { "_attr" :
               { "_orgname" : "MNM-Team",
                 "_type" : "datacentre",
                 "_location" : "Oettingenstra e 67",
                 "_contact" : "abuse@mnm-team.org"
             },
             "_sons" :
               {"virt-lab" :
                   { "_attr" :
                       { "_type" : "ou",
                         "_contact" : "abuse@mnm-team.org"
                       },
                     "_sons" :
                       { "RNP" :
                           { "_attr" :
                             "_type" : "cluster",
                             "_tech" : "xen",
                             "_version" : "4.4",
                             "_config" : "cluster.conf"
                           }
                       }
                   }
               }
           }
       }
   }
}
```

Ein Neustart von VIFS erzeugt die neue Basisdomänenstruktur und die Befehle

“echo managed >/mnt/VIFS/MNM/barraca/virt-lab/RNP/Gruppe01”

sowie

“echo managed >/mnt/VIFS/MNM/barraca/virt-lab/RNP/Gruppe02”

laden die gewünschten Strukturen in VIFS und erzeugen weitere Verzeichnisse und Dateien. Um dem RNP-Administrator ausreichende Verwaltungsmöglichkeiten zur Verfügung zu stellen, wurden neue Domänen in LDAP erzeugt (Abbildung 6.2) und die LDAP-ACL modifiziert (Auflistung 6.3). Anschließend übergaben die VIFS-Administratoren auf Anforderung eines MNM-Administrators den Besitz des RNP Verzeichnisses sowie alle enthaltenen Elemente an den Nutzer “rnpadmin01” und die Gruppe “rnp”.

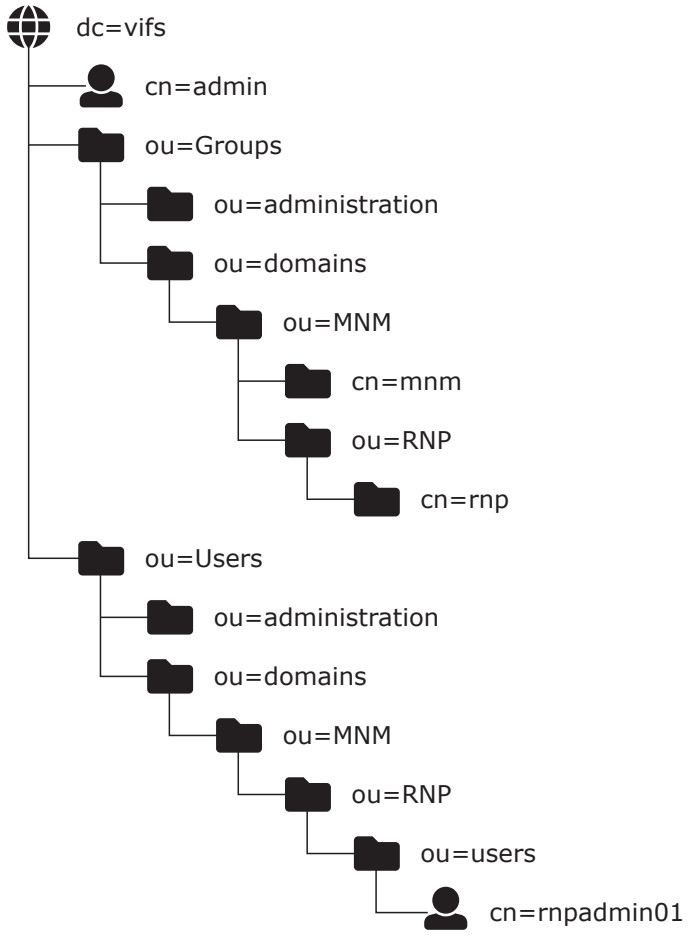

Abbildung 6.2: RNP-Domäne

Listing 6.3: Erweiterte ACL

```
olcAccess: to dn.subtree="ou=RNP,ou=MNM,ou=domains,ou=Groups,dc=vifs"
   by dn.subtree="ou=users,ou=RNP,ou=MNM,ou=domains,ou=Users,dc=vifs"
      manage
   by dn.subtree="cn=rnp,ou=RNP,ou=MNM,ou=domains,ou=Groups,dc=vifs"
      manage
   by * read
olcAccess: to dn.subtree="ou=RNP,ou=MNM,ou=domains,ou=Users,dc=vifs"
   by dn.subtree="ou=users,ou=RNP,ou=MNM,ou=domains,ou=Users,dc=vifs"
      manage
   by dn.subtree="cn=rnp,ou=RNP,ou=MNM,ou=domains,ou=Groups,dc=vifs"
      manage
   by * read
olcAccess: to attrs=userPassword,shadowLastChange
    by self write
    by dn.subtree="ou=users,ou=administration,ou=Users,dc=vifs"
       write
    by anonymous auth
    by * none
olcAccess: to *
    by dn.subtree="ou=users,ou=administration,ou=Users,dc=vifs"
       manage
    by dn.subtree="cn=vifsadmin,ou=administration,ou=Groups,dc=vifs"
       manage
    by * read
```

6.4 Autarke Domänen

Alle folgenden Operationen wurden unter dem Benutzer “rnpadmin01” ausgeführt.

Der RNP-Administrator verfügt über die gewünschte VIFS-Anbindung der Infrastruktur und erzeugt im nächsten Schritt die benötigten Gruppen sowie Benutzer über das LDAP-Frontend innerhalb der ihm zugewiesenen Domänen (ou=RNP, ou=MNM, ou=domains, ou=Groups, dc=vifs & ou=RNP, ou=MNM, ou=domains, ou=Users, dc=vifs).
Hierbei wurden, um Übersicht zu wahren, zwei Studentengruppen zu jeweils zwei Studenten, sowie ein RNP-Betreuer erzeugt und entsprechenden Gruppen zugewiesen. Um bei der Zuweisung der entsprechenden Gruppen keinen Eingriff seitens der VIFS-Administratoren zu erfordern, fügte der “rnpdamin01” sich selbst zu allen erzeugten Gruppen hinzu. Durch dieses Vorgehen kann der RNP-Administrator die Gruppen im Dateisystem entsprechend zuteilen und so den Studenten Zugriff auf die Laborumgebungselemente erteilen. Ob der RNP-Administrator sich selbst aus diesen Gruppen wieder entfernt, bleibt ihm überlassen. Er kann sich schließlich jederzeit wieder zu den erstellten Gruppen hinzufügen, es sei denn die ACL-Struktur wird verändert. Die für das RNP erzeugte Domänenstruktur ist in Abbildung 6.3 zu sehen.

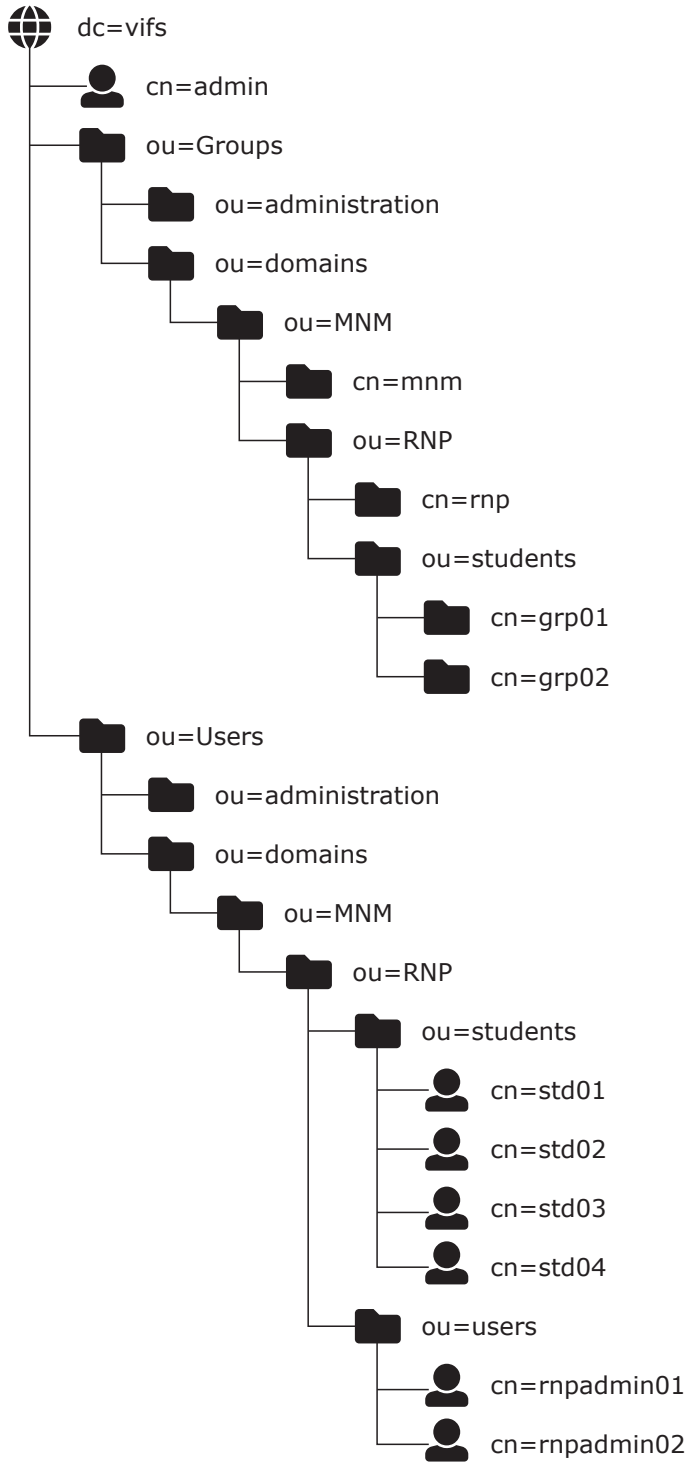

Abbildung 6.3: Erzeugte RNP-Domänenstruktur

Im Dateisystem wird durch die ausgeführten Aktionen die in Abbildung 6.4 gezeigte Rechtestruktur erzeugt.

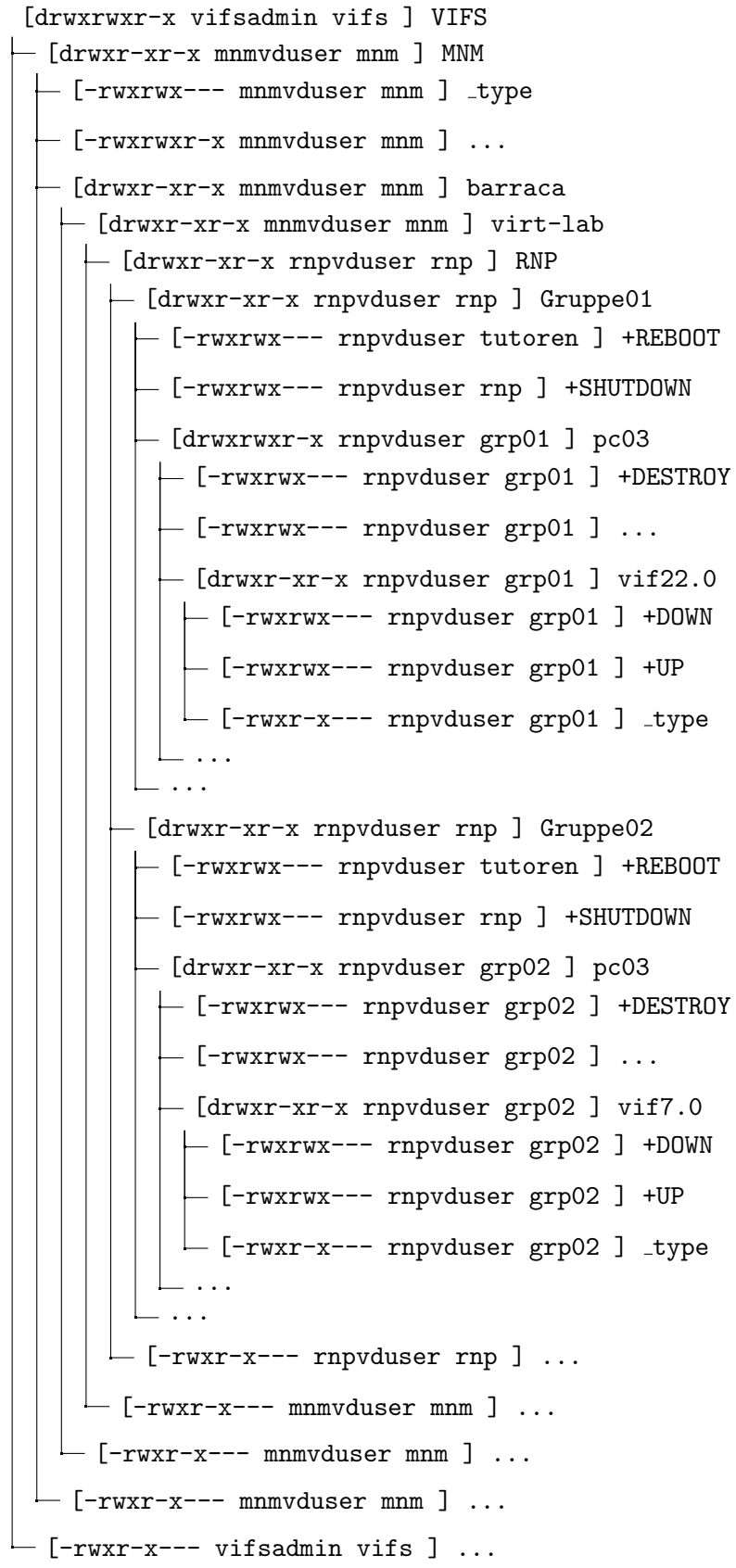

Abbildung 6.4: Zugriffsrechte im Dateisystem

6.5 Mögliche Optimierungen und Zusammenfassung

Der erzeugte Prototyp bietet die Möglichkeit das Management des RNP über VIFS zu betreiben. Hierbei können Domänen weitestgehend autark agieren und sich ihre Umgebung bezüglich Zugriffsrechten und Nutzerverwaltung nach belieben gestalten. Einzig die Besitzübergabe von Objekten bedarf eines Eingriffs eines systemweiten VIFS-Administrators. Die Erweiterung der VIFS-Struktur um zusätzliche Domänen ist möglich und eine persistente Speicherung der Zugriffsrechte ist vorhanden. Eine detaillierte Evaluation des Prototyps könnte im Zuge eines Einsatzes als Zugriffsalternative für alle RNP-Beteiligten im kommenden Wintersemester 2018/19 durchgeführt werden.

Die Implementierung eines sicheren Prüfverfahrens, welches den Zugriff auf die Besitzübergabeoperationen erlaubt, könnte die Selbstständigkeit und Unabhängigkeit der Verwaltungsdomänen zusätzlich erhöhen. Weiters wäre ein Hilfsmittel zur einfacheren Verwaltung der LDAP-ACL wünschenswert, um leider häufig auftretende Fehlkonfigurationen zu reduzieren und die Übersichtlichkeit zu erhöhen.

7 Zusammenfassung und Ausblick

Durch die heterogene Natur von rechnerbasierten Infrastrukturen entstehen hohe Anforderungen an ein Rechtemodell, welches eine Abbildung dieser Strukturen - und vor allem der zugehörigen Rechtekonstrukte - in ein Dateisystem ermöglichen soll. Diese Anforderungen wurden unter Beachtung der POSIX-Konformität sowie dem Aufrechterhalten der Dateisystemmetapher erfüllt. Durch die ausführliche Beschreibung des RNP wurde eine verschachtelte Infrastruktur dargestellt und anschließend in VIFS abgebildet. Durch die Zuhilfenahme von LDAP wurde die relativ unflexible traditionelle POSIX-Nutzerverwaltung an ein flexibles, POSIX-konformes und ACL-unterstützendes System ausgelagert. Dieses Vorgehen ermöglichte die Erzeugung eines Prototypen, der alle eruierten Anforderungen erfüllt und nahezu autarke Domänen ermöglicht. Die Grenzen der Skalierbarkeit sowie die Praxistauglichkeit dieses Prototypen müssen nun evaluiert werden. In dem Szenario, das für diese Arbeit verwendet wurde, war es möglich eine sehr schwache Zugangspolitik zu nutzen und jedem Nutzer Zugang zu jedem beliebigen VIFS-Verzeichnis zu erlauben. Restriktivere Zugangskonventionen könnten im vorhandenen Prototyp bereits mittels "Barriereverzeichnissen" einfach umgesetzt werden. Diese Barriereverzeichnisse könnten durch vorhandene Objektverzeichnisse oder durch zusätzliche Verzeichnisse realisiert werden.

Werden Organisationsdomänen so granular gewählt, dass sie den gewünschten Rollen entsprechen, ist eine "eins-zu-eins" Abbildung von Rollen in POSIX möglich. Dies gilt jedoch nur, falls keine von verschiedenen Rollen gemeinsam genutzten Ressourcen vorkommen. Die Auflösung einer solchen Situation erfordert eine zusätzliche Benutzergruppe für das entsprechende Objekt, was die Verwaltungskomplexität erhöht und die "eins-zu-eins"-Abbildung von Rollen zu Benutzergruppen zerstört. In der weiteren Entwicklung wird sich die Frage stellen, ob die Aufrechterhaltung der Dateisystemmetapher sowie die Verwendung der gewählten FUSE-Variante den Verzicht auf flexiblere Rollenkonstrukte rechtfertigt.

Eine äußerst sicherheitskritische, aber unter Umständen notwendige künftige Erweiterung, wird die Implementierung der Berechtigungsprüfung zur Nutzung von Programmen welche erhöhte Zugriffsrechte benötigen (chown) sein, um so vollständig autarke Domänen zu ermöglichen. Viele Betriebssysteme zeigen bereits anhand verschiedener Programme (passwd, chfn, u.A.) wie eine objektgebundene Zugriffsrechteerweiterung durch Nutzung des SUID-Bits realisiert werden könnte. Hierbei müsste jedoch eine weit umfangreichere Prüfung stattfinden, um zusätzlich zur vom Besitzer initiierten Übergabe auch eine von übergeordneten Domänen initiierte Übergabe beziehungsweise ein "Zurückholen" von Elementen zu ermöglichen.

Abschließend sei erwähnt, dass die Verwendung von VIFS - auch unter Einsatz des vorliegenden Rechtemodells - stets auf einem zugrundeliegenden Vertrauensverhältnis gegenüber dem VIFS-Betreiber zu basieren hat, da eine Manipulation der Rechtestruktur seitens des Betreibers nicht ausgeschlossen werden kann.

Anhang

1 API

Die hier dargestellte API ist keineswegs vollständig. Sie soll die Komplexität des heterogenen Umfelds, der benötigten Parameter sowie der erforderlichen Zugriffsrechte verdeutlichen. Die Operation "open_terminal" stellt hierbei eine Alternative für alle unter Umständen noch benötigten Operationen dar. Hierbei wird davon ausgegangen, dass diese Operation uneingeschränkten Zugriff auf das betroffene Objekt zur Verfügung stellt. Schwerpunkt dieser API ist das Domänenobjekt. Man beachte hierbei die überladenen Delegationsmethoden.

```
object {
  boolean read_att(obj, att)
  /** If:   executing user has no rights for executing this
  *         operation on either the object or the attribute,
  *         the object or the attribute does not exist
  *   then: the function will return false
  *   else: the attribute will be shown and true will be returned
  */

  boolean modify_att(obj, att)
  /** If:   executing user has no rights for executing this
  *         operation on either the object or the attribute,
  *         the object or the attribute does not exist
  *   then: the function will return false
  *   else: the attribute will be openend with write permissions
  *         in an editor and true will be returned
  */
}

bootable_object extends object {
  boolean reboot(obj)
  /** If:   executing user has no rights for executing this
  *         operation on the object,
  *         the object does not exist,
  *         the object is not running
  *   then: the function will return false
  *   else: the object will be immediately rebooted and true
  *         will be returned
  */

  boolean shutdown(obj)
  /** If:   executing user has no rights for executing this
  *         operation on the object,
  *         the object does not exist,
  *         the object is not running
```

```
    *   then: the function will return false
    *   else: the object will be immediately shutdown and
    *         true will be returned
    */

    boolean start(obj)
    /** If:   executing user has no rights for executing this
    *         operation on the object,
    *         the object does not exist,
    *         the object is running,
    *   then: the function will return false
    *   else: the object will be started and true will be returned
    */
  }

  virtualizing_object extends bootable_object{
    boolean create_vm(obj-host, obj, conf)
    /** If:   executing user has no rights for executing this
    *         operation on the object,
    *         the object-host does not exist,
    *         the object-host is not running,
    *         the obj-host does not support virtualization,
    *         the configuration-file is not readable or does not exist,
    *         the amout of RAM or Disk-Memory inquired by the
    *         configuration is not available,
    *         the number of processor inquired by
    *         configuration is not available,
    *         the number of sockets inquired by
    *         configuration is not available
    *   then: the function will return false
    *   else: a vm based on the values of the configuration file *conf*
    *         will be created and true will be returned
    */

    boolean delete_vm(obj-host, obj)
    /** If:   executing user has no rights for executing this operation
    *         on the object,
    *         the object-host does not exist,
    *         the object-host is not running,
    *         the object is not running on the object-host,
    *         the object is not shutdown,
    *   then: the function will return false
    *   else: a terminal on the object will be opened (as user?) and
    *         true will be returned
    */

    boolean create_bridge(bridge-host, bridge-name)
    /** If:   executing user has no rights for executing this operation
    *         on the object,
    *         the bridge-host does not exist,
    *         the bridge-host is not running,
    *         the bridge-name is already in use,
    *   then: the function will return false
    *   else: a bridge named like *bridge-name* will be created and
    *         true will be returned
    */
```

```
boolean append_iface(bridge-host, bridge-name, iface)
/** If:   executing user has no rights for executing this operation
*         on the object,
*         the bridge-host does not exist,
*         the bridge-host is not running,
*         the interface *iface* does not exist,
*   then: the function will return false
*   else: the interface *iface* will be appended to the bridge and
*         true will be returned
*/

boolean remove_iface(bridge-host, bridge-name, iface)
/** If:   executing user has no rights for executing this operation
*         on the object,
*         the bridge-host does not exist,
*         the bridge-host is not running,
*         the interface *iface* does not exist,
*         the interface *iface* is not appended to the bridge,
*   then: the function will return false
*   else: the interface *iface* will be removed from the bridge and
*         true will be returned
*/

boolean delete_bridge(bridge-host, bridge-name)
/** If:   executing user has no rights for executing this operation
*         on the object,
*         the bridge-host does not exist,
*         the bridge-host is not running,
*         the bridge-name does not exist,
*   then: the function will return false
*   else: the bridge will be deleted and true will be returned
*/

boolean migrate(host, obj, target)
/** If:   executing user has no rights for executing this operation
*         on the host,
*         executing user has no rights for executing create_vm on target,
*         executing user has no rights for executing delete_vm on host,
*         the obj does not exist,
*         the target does not exist or is not running,
*   then: the function will return false
*   else: the virtualized object will be migrated to the target and
*         true will be returned
*/

boolean link_vms(host, obj1, iface1, obj2, iface2)
/** If:   executing user has no rights for executing this operation
*         on the host,
*         executing user has no rights for executing manipulations
*         of iface1 on obj1
*         executing user has no rights for executing manipulations
*         of iface2 on obj2
*         obj1 or obj2 does not exist,
*         obj1 or obj2 is not running,
*         iface1 or iface2 does not exist,
*   then: the function will return false
*   else: the virtualized objects will be connected and true will
```

```
    *         be returned
    */

    boolean unlink_vms(host obj1, iface1, obj2, iface2)
    /** If:   executing user has no rights for executing this operation
    *         on the host,
    *         executing user has no rights for executing manipulations
    *         of iface1 on obj1
    *         executing user has no rights for executing manipulations
    *         of iface2 on obj2
    *         obj1 or obj2 does not exist,
    *         obj1 or obj2 is not running,
    *         iface1 or iface2 does not exist,
    *   then: the function will return false
    *   else: the virtualized objects will be disconnected and true will
    *         be returned
    */

    boolean suspend(obj)
    /** If:   executing user has no rights for executing this
    *         operation on the object,
    *         the object does not exist,
    *         the object is not running,
    *   then: the function will return false
    *   else: the object will be suspended and true will be returned
    */

    boolean resume(obj)
    /** If:   executing user has no rights for executing this
    *         operation on the object,
    *         the object does not exist,
    *         the object is not in suspend-mode,
    *   then: the function will return false
    *   else: the object will be unsuspended and true will be returned
    */
  }

  virtualized_object extends virtualizing_object{
    boolean pause(host, obj)
    /** If:   executing user has no rights for executing this operation
    *         on the host for the object,
    *         the obj does not exist,
    *         the obj is not running,
    *         the obj is paused,
    *   then: the function will return false
    *   else: the virtualized object will be paused and true will
    *         be returned
    */

    boolean unpause(host, obj)
    /** If:   executing user has no rights for executing this operation
    *         on the object,
    *         the obj does not exist,
    *         the obj is not paused,
    *   then: the function will return false
    *   else: the virtualized object will be unpaused and true will
    *         be returned
```

```
  */

  boolean destroy(host, obj)
  /** If:   executing user has no rights for executing this operation
  *         on the object,
  *         the obj does not exist,
  *         the obj is not running,
  *   then: the function will return false
  *   else: the virtualized object will be destroyed (killed) and
  *         true will be returned
  */
}

physical_machine extends virtualizing_object{
  boolean open_terminal(obj)
  /** If:   executing user has no rights for executing this operation
  *         the the object,
  *         the object does not exist,
  *         the object is not running,
  *   then: the function will return false
  *   else: a terminal on the object will be opened (as user?)
  *         and true will be returned
  */
}


virtual_machine extends virtualized_object{
  boolean open_terminal(obj)
  /** If:   executing user has no rights for executing this operation
  *         on the object,
  *         the object does not exist,
  *         the object is not running,
  *   then: the function will return false
  *   else: a terminal on the object will be opened and true
  *         will be returned
  */
}

iface extends object{
  boolean ifdown(obj, iface)
  /** If:   executing user has no rights for executing this operation
  *         on the object,
  *         the object does not exist,
  *         the object is not running,
  *         the iface does not exist,
  *         the ifacestate is not "up",
  *   then: the function will return false
  *   else: the ifacestate of the object will be set to "down"
  *         and true will be returned
  */

  boolean ifup(obj, iface)
  /** If:   executing user has no rights for executing this operation
  *         on the object,
  *         the object does not exist,
  *         the object is not running,
  *         the iface does not exist,
```

```
    *          the ifacestate is not "down",
    *   then: the function will return false
    *   else: the lifacestate of the object will be set to "up" and
    *          true will be returned
    */
  }

  domain extends object{
    boolean create_user(domain, user)
    /** If:   executing user has no rights for executing this operation
    *          on the specific domain,
    *          the user already exists in domain,
    *   then: the function will return false
    *   else: a user will be created on every object belonging to the
    *          domain and true will be returned
    */

    boolean grant_permission(domain, user, obj, op)
    /** If:   executing user has no rights for executing this operation,
    *          executing user has no delegation-rights to the
    *          object-bound operation,
    *          the target-user does not exist,
    *          the object does not exist,
    *          the operation does not exist on object,
    *   then: the function will return false
    *   else: the executable operation bound to the object will
    *          be granted to the target-user and true will be returned
    */

    boolean grant_delegation_permission(domain, user, obj, op)
    /** If:   executing user has no rights for executing this operation,
    *          executing user has no delegation-rights to the
    *          object-bound operation,
    *          the target-user does not exist,
    *          the object does not exist,
    *          the operation does not exist on object,
    *   then: the function will return false
    *   else: the delegatable operation bound to the object will
    *          be granted to the target-user and true will be returned
    */

    boolean revoke_permission(domain, user, obj, op)
    /** If:   executing user has no rights for executing this operation,
    *          executing user has no delegation-rights to the
    *          object-bound operation,
    *          the target-user does not exist,
    *          the object does not exist,
    *          the operation does not exist on object,
    *   then: the function will return false
    *   else: the executable or delegatable operation bound to the object
    *          will be revoked from the target-user and true will be returned
    */

    boolean delete_user(domain, user)
    /** If:   executing user has no rights for executing this operation
    *          on the specific domain,
    *          the user does not exist in domain,
```

```
*   then: the function will return false
*   else: the user will be deleted on every object belonging to the
*         domain and true will be returned
*/

boolean create_role(domain, role)
/** If:   executing user has no rights for executing this operation
*         on the domain,
*         the role already exists in domain,
*   then: the function will return false
*   else: the role will be created for the domain and
*         true will be returned
*/

boolean grant_permission(domain, role, obj, op)
/** If:   executing user has no rights for executing this operation
*         on the domain,
*         executing user has no delegation-rights to the
*         object-bound operation,
*         the role does not exist in domain,
*         the object does not exist,
*         the operation does not exist on object,
*   then: the function will return false
*   else: the executble operation bound to the object/domain will
*         be added to the role and true will be returned
*/

boolean grant_delegation_permission(domain, role, obj, op)
/** If:   executing user has no rights for executing this operation,
*         executing user has no delegation-rights for the
*         object-bound operation,
*         the role does not exist in domain,
*         the object/domain does not exist,
*         the operation does not exist on object,
*   then: the function will return false
*   else: the delegatable operation bound to the object/domain will
*         be added to the role and true will be returned
*/

boolean revoke_permission(domain, role, obj, op)
/** If:   executing user has no rights for executing this operation,
*         executing user has no delegation-rights for the
*         object-bound operation,
*         the role does not exist in domain,
*         the object does not exist,
*         the operation does not exist on object,
*   then: the function will return false
*   else: the operation bound to the object/domain will be removed
*         from the role and true will be returned
*/

boolean assign_role(domain, user, role)
/** If:   executing user has no rights for executing this operation,
*         the role does not exist in domain,
*         the role is already assigned to the target-user,
*   then: the function will return false
*   else: the role will be assigned to the user and
```

```
    *          true will be returned
    */

    boolean unassign_role(domain, user, role)
    /** If:   executing user has no rights for executing this operation,
    *          the role does not exist in domain,
    *          the role is not assigned to the target-user,
    *   then: the function will return false
    *   else: the role will be unassigned from the user and
    *          true will be returned
    */

    boolean delete_role(domain, role)
    /** If:   executing user has no rights for executing this operation,
    *          the role does not exists in domain,
    *   then: the function will return false
    *   else: the role will be deleted from the domain and
    *          true will be returned
    */

    boolean create_domain(parent-domain, domain)
    /** If:   executing user has no rights for executing this operation,
    *          the domain already exists in parent-domain,
    *   then: the function will return false
    *   else: the domain will be created for in the domain and
    *          true will be returned
    */

    boolean delete_domain(parent-domain, domain)
    /** If:   executing user has no rights for executing this operation,
    *          the domain is not empty,
    *          the domain does not exist in parent-domain,
    *   then: the function will return false
    *   else: the role will be created from domain and
    *          true will be returned
    */

  boolean join_domain(domain, obj)
    /** If:   executing user has no rights for executing this operation,
    *          the object does already exist in domain,
    *   then: the function will return false
    *   else: the object will be added to the target-domain and
    *          true will be returned
    */

    boolean remove_from_domain(domain, obj)
    /** If:   executing user has no rights for executing this operation,
    *          the object does not exist in domain,
    *   then: the function will return false
    *   else: the object will be removed from the target-domain and
    *          true will be returned
    */
  }
```

2 CODE

2.1 Datenbankverwaltungsmodul

Das in Listing 7.1 gezeigte Programm übernimmt die Kommunikation mit der SQLite-Datenbank, welche sich im VIFS-Verzeichnis befindet.

Listing 7.1: struct_manager.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
__author__='Gerhard Groeschl'
__doc__="""Programm for handling persistent rightstructure"""
import sqlite3 as sql
from constants import *


def persist(_path, _user, _group, _mod, att):

  con = sql.connect('structure.db')

  if _user==-1: _user=getuid(_path);
  if _group==-1: _group=getgid(_path);
  if _mod==-1: _mod=getmod(_path,att);

  if _user==None: _user=0;
  if _group==None: _group=0;
  if _mod==None: _mod=DEFAULT_MODE_DIR;

  with con:
    cur = con.cursor()
    cur.execute(
                "CREATE TABLE IF NOT EXISTS structure"
                "(__path TEXT PRIMARY KEY, __user INT,"
                "_group INT, __mod INT);"
                )

    check_str = "SELECT __path FROM structure WHERE __path = "
                 + "\"" + _path + "\"" + ";"

    cur.execute(check_str)

    row = cur.fetchone()
    if row == None:
         ex_str = "INSERT INTO structure VALUES(" + "\""
                  + _path + "\"" + "," + str(_user) + ","
                  + str(_group) + "," + str(_mod) + ");"
         cur.execute(ex_str)
```

```
40      else:
41          cur.execute(
42                      "UPDATE structure SET  user=?,  group=?,"
43                      " mod=? WHERE  path=?",
44                      (str(_user),
45                      str(_group),
46                      str(_mod),
47                      _path )
48                      )
49      con.commit()
50
51 def getuid(key):
52
53     con = sql.connect('structure.db')
54     with con:
55       cur = con.cursor()
56       cur.execute(
57                      "CREATE TABLE IF NOT EXISTS structure "
58                      "(  path TEXT PRIMARY KEY,  user INT, "
59                      " group INT,  mod INT);"
60                      )
61
62       cur.execute(
63                      "SELECT  user from structure WHERE  path="
64                      + "\"" + str(key) +"\"" + ";"
65                      )
66
67       row = cur.fetchone()
68       if row == None:
69           return 0
70       else:
71           row = str(row)
72           row = row[1:-2]
73           return int(row)
74
75 def getgid(key):
76     con = sql.connect('structure.db')
77     with con:
78
79       cur = con.cursor()
80       cur.execute(
81                      "CREATE TABLE IF NOT EXISTS structure "
82                      "(  path TEXT PRIMARY KEY,  user INT, "
83                      " group INT,  mod INT);"
84                      )
85       cur.execute(
86                      "SELECT  group from structure WHERE  path="
```

```
87                      + "\"" + str(key) +"\"" + ";"
88                      )
89      row = cur.fetchone()
90      if row == None:
91          return 0
92      else:
93          row = str(row)
94          row = row[1:-2]
95          return int(row)
96
97 def getmod(key, att):
98     con = sql.connect('structure.db')
99     with con:
100
101       cur = con.cursor()
102       cur.execute(
103                       "CREATE TABLE IF NOT EXISTS structure "
104                       "(  path TEXT PRIMARY KEY,  user INT, "
105                       " group INT,  mod INT);"
106                       )
107       cur.execute(
108                       "SELECT  mod from structure WHERE  path="
109                       + "\"" + str(key) +"\"" + ";"
110                       )
111       row = cur.fetchone()
112       if row == None:
113          if att=='dir':
114           return DEFAULT_MODE_DIR
115          if att=='file':
116           return DEFAULT_MODE_FILE
117
118       else:
119           row = str(row)
120           row = row[1:-2]
121           return int(row)
```

2.2 Modifikationen an bestehendem VIFS-Code

In der Hauptdatei von VIFS mussten geringfügige Veränderungen vorgenommen werden, um die Speicherung der vergebenen Rechte zu ermöglichen und zeitgleich in die Datenbank zu schreiben.

Listing 7.2: VIFS.py

```
1   __author__='Vitalian A. Danciu'
2
3   ...
4
5   def chmod(self, path, mode):
6     info("chmod %s to mode %o"%(path,mode))
7     if not self.fstree.exists(path) :
8           raise FuseOSError(ENOENT)
9     if self.fstree.gettype(path) == SYMLINKTYPE:
10      effpath = _deref_symlink_recurse(path)
11    else:
12      effpath = path
13    attrval = self.fstree.get_attribute(effpath, 'st_mode')
14    attrval &= 0770000
15    attrval |= mode
16    """
17    checking object-type (dir/file) to choose which
18    standard-attributes are used if no entries in
19    database exist
20    """
21    if self.fstree.gettype(path) == DIRTYPE:
22        struct.persist(effpath,-1,-1,mode, 'dir')
23    if self.fstree.gettype(path) == FILETYPE:
24        struct.persist(effpath,-1,-1,mode, 'file')
25
26    self.fstree.set_attribute(effpath, 'st_mode', attrval)
27    self._update_mtime(effpath, time())
28    return 0
29
30  def chown(self, path, uid, gid):
31    info("chown %s to gid %s and uid %s"%(path,gid,uid))
32    if not self.fstree.exists(path) :
33        raise FuseOSError(ENOENT)
34    if self.fstree.gettype(path) == SYMLINKTYPE:
35      effpath = self._deref_symlink_recurse(path)
36    else:
37      effpath = path
38    if uid != -1:
39        self.fstree.set_attribute(effpath, 'st_uid', uid)
40    if gid != -1:
```

```
41             self.fstree.set_attribute(effpath, 'st_gid', gid)
42         struct.persist(effpath, uid, gid, -1, '_')
43
44         ...
```

Die Datei fstorage.py regelt die Erzeugung von Dateisystemelementen. Hierbei wurde die Wahl des Besitzers eines neu erzeugten Objekts so manipuliert, dass diese stets dem Besitzer der beinhaltenden Domäne, also dem beinhaltenden Verzeichnis, zugeordnet wird. Die zugehörigen Rechte werden durch vorgegebene Standardwerte, abhängig von der Art des Dateisystemelements (Datei oder Verzeichnis) gewählt.

Listing 7.3: fstorage.py

```
1   __author__='Vitalian A. Danciu'
2
3   ...
4
5   def mkdir(self, fatherpath, newdir, mode):
6    """Add a directory. Caution: assumes,
7       that input has been validated.
8       fatherpath: the parent directory
9       newdir: the name of the new directory
10      mode: the file mode, e.g. 0775 for rwxrwxr-x
11   """
12   debug("%s: %s; %s; %s."
13        %(giveupthefunc(),fatherpath,newdir,str(mode)))
14
15   fref = self.getref(fatherpath)
16   if fref == None: raise NoSuchPath(path)
17   fref[SONS][newdir] = { TYPE : DIRTYPE, SONS : {} }
18
19   """
20   if entry-values in db exists -> take it
21   otherwise use owner, group and mod of fatherpath
22   for new element. in worst case take root rights
23   """
24   if struct.getuid(fatherpath + "/" + newdir)==0:
25       uid=struct.getuid(fatherpath)
26   else:
27       uid = struct.getuid(fatherpath + "/" + newdir)
28   if struct.getgid(fatherpath + "/" + newdir)==0:
29       gid = struct.getgid(fatherpath)
30   else:
31       gid = struct.getgid(fatherpath + "/" + newdir)
32
33   mod = struct.getmod(fatherpath + "/" + newdir, 'dir')
34   struct.persist(
35            fatherpath + "/" + newdir,
36            uid, gid,
37            mod, 'dir')
38   fref[SONS][newdir][ATTR] = dir_initdict(
39                mod,uid,gid,
40                self.get_inode(subname(fatherpath, newdir))
```

```
41                 )
42
43
44
45
46  def create(self, path, newfile, mode):
47    """Create a file.
48       path: the path to the file, including the file's name
49       newfile: the name of the file (without path)
50       mode: the file mode, e.g. 0664 for rw-rw-r--
51    """
52    fref = self.getref(father_path(path))
53    fref[SONS][newfile] = {TYPE:FILETYPE}
54    fref[SONS][newfile][CONTENT] = ''
55    fatherpath=father_path(path)
56    if struct.getuid(path)==0:
57        uid=struct.getuid(fatherpath)
58    else:
59        uid = struct.getuid(path)
60    if struct.getgid(path)==0:
61        gid = struct.getgid(fatherpath)
62    else:
63        gid = struct.getgid(path)
64    mod = struct.getmod(fatherpath,'file')
65    struct.persist(path, uid, gid, mod, 'file')
66    fref[SONS][newfile][ATTR] =
67            file_initdict(mod,uid,gid, self.get_inode(path))
68    debug("Type of %s is %s"%(path, self.gettype(path)))
69    return self.new_fd(path)
70
71    ...
72
73  def put_file_data(self, path, data):
74    """Create a file at path and fill it with data,
75       without triggering hooks.
76    """
77    content = bytes(data)
78    now = time()
79    newfile = entry_name(path)
80    d = self.fatherref(path)
81    d[SONS][newfile] = dict()
82    fileref = d[SONS][newfile]
83    fileref[TYPE] = FILETYPE
84    fileref[CONTENT] = content
85    fatherpath=father_path(path)
86    if struct.getuid(path)==0:
87        uid=struct.getuid(fatherpath)
```

```
88      else:
89          uid = struct.getuid(path)
90      if struct.getgid(path)==0:
91          gid = struct.getgid(fatherpath)
92      else:
93          gid = struct.getgid(path)
94      mod = struct.getmod(path,'file')
95
96      fileref[ATTR]= dict(
97                  st_mode = (mod | S_IFREG), st_nlink=1,
98                  st_size=len(content), st_ctime=now,
99                  st_mtime=now, st_atime=now, st_uid=uid,
100                 st_gid=gid, st_ino=self.get_inode(path))
101
102     ...
```

Abbildungsverzeichnis

Literaturverzeichnis

[acl03] Access Control Lists in Linux. (2003). http://www.cs.unibo.it/~renzo/doc/papers/AccessControlListInLinux.pdf

[APST05] Anderson, Thomas ; Peterson, Larry ; Shenker, Scott ; Turner, Jonathan: Overcoming the Internet impasse through virtualization. In: *Computer* 38 (2005), Nr. 4, S. 34–41

[Dan16a] Danciu, Kranzlmüller Guggemos: Schichtung virtueller Maschinen zu Labor- und Lehrinfrastruktur. In: *DFN-Forum Kommunikationstechnologien* 9 (2016), S. 35–45

[Dan16b] Danciu, Vitalian A.: A file-system abstraction for virtualized infrastructure. (2016)

[fus] Filesystem in Userspace. http://man7.org/linux/man-pages/man8/mount.fuse.8.html

[HGH] Heinz Gerd Hegering, Bernhard N. Sebastian Albeck A. Sebastian Albeck: Integriertes Management vernetzter Systeme.

[IG] IEEE, The ; Group, The O.: The OSI network management model.

[kvm] KVM-documentation. http://www.qemu.org/documentation/

[posa] The Open Group Base Specifications. http://pubs.opengroup.org/onlinepubs/9699919799/

[posb] The Open Group Base Specifications - Character Set. http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap06.html#tagtcjh_3

[rnp] Rechnernetze Praktikum. http://www.nm.ifi.lmu.de/teaching/Praktika/2016ws/rnp/

[xen] XEN-documentation. https://www.xenproject.org/help/documentation.html