Dokumentation
über das

Backend

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Zielsetzung . . . 3
1.2 Forderungen an das System . . . 3
1.3 Pfade von VAMOS am DESY-Zeuthen . . . 4

2 Die Datenbankstruktur **4**
2.1 Forderungen . . . 4
2.2 Struktur der VAMOS Datenbank . . . 6
2.3 Erklärung der Datenbankstruktur . . . 7

3 Die Repräsentation der Daten im Speicher **8**
3.1 Überlegungen . . . 8
3.2 Beispiel . . . 10
3.3 Erklärung der Datenstruktur . . . 11

4 Das Perl-Interface zur Datenbank **11**
4.1 Übersicht . . . 11
4.2 Db.pm . . . 12
4.3 TopObject.pm . . . 13
4.4 Tables.pm . . . 13
4.5 Include_Object.pm . . . 15
4.6 SynCheck.pm . . . 15
4.7 VamosLog.pm . . . 16
4.8 ConfigVars.pm . . . 16
4.9 Possible_Values.pm . . . 16

5 Die Client / Server Architektur **18**
5.1 Der VAMOS-Server . . . 18
5.1.1 Allgemeine Probleme . . . 19
5.1.2 Konfiguration . . . 20
5.1.3 Der Loginprozeß . . . 20
5.1.4 Datensynchronisation zwischen den Serverprozessen . . . 21
5.2 Das Client-Modul . . . 22
5.3 Sicherheit . . . 22
5.3.1 Verschlüsselung . . . 22
5.3.2 Zugriffsschutz / ACLs . . . 22
5.4 Transparenz beim Datenzugriff . . . 23

6 Dateigenerierung **23**

7 Bekannte Probleme **24**
7.1 Der VAMOS Server 24
7.2 Die MySQL Datenbank 24

8 Abschließende Bemerkungen **24**

9 Anhang **25**
9.1 Benötigte Perl-CPAN-Module 25

1 Einleitung

Dies ist eine Übersicht, die das Zusammenspiel der verschiedenen Module des VAMOS Backends zeigen soll. In den einzelnen Modulen befinden sich Detailbeschreibungen als Kommentare, sowie größtenteils Pod-Dokumentationen[1], die über *perldoc <Modulname>* abgerufen werden können.

1.1 Zielsetzung

Ziel des VAMOS Projektes war es, das veraltete Systemadministrationstool GenuAdmin zu ersetzen. Es basierte auf reinen ASCII Dateien und schränkte somit die Administrationsmöglichkeiten ein. So kam es z.B. vor, daß mehrere Nutzer gleichzeitig die Datei editierten so somit die Änderungen der anderen zu nichte machten.

Es sollte ein neues Werkzeug geschaffen werden, das auf Grundlage einer relationalen Datenbank arbeiten muß. Dieses Werkzeug sollte die Daten auf der Datenbank in einer leicht bedienbaren Form anzeigen und modifizieren können. Daraus erwuchs die Idee alle Daten, die über Fremdschlüssel auf der Datenbank verbunden sind, zu Objekten zusammenzufassen und sie so darzustellen.

Daher rührt auch der Name: VAMOS – A **V**ersatile **A**dminitstration tool for **M**ultiple **O**perating **S**ystems.

1.2 Forderungen an das System

Eine Forderung an das Programm war die leichte Abfrage und Modifikation der Daten aus Skripten heraus. VAMOS sollte unter UNIX Betriebssystemen laufen (Linux, Solaris, HP-UX). Als Option sollte die Verwendung unter Windows (NT) aber nicht grundsätzlich ausgeschlossen werden. Die besten Voraussetzungen dafür bot die Skriptsprache Perl. Für Perl sprachen auch die vielen bereits vorgefertigten Module, die jedem Programmierer zur Verfügung stehen. Die Anbindung an (fast) beliebige Datenbanken sowie spätere mögliche Features (Client-Server Mechanismus, Verschlüsselung, Kerberos Authentisierung, ...) standen in jedem Fall für eine Aufnahme in das Programm bereit.

Die zugrunde liegende relationale Datenbank sollte nicht fest vorgegeben sein. Letztendlich müßte das Programm mit jeder arbeiten können. Somit mußte sichergestellt werden, daß der Zugriff ausschließlich über ANSI-SQL zu geschehen hatte. Befehle die nur auf einigen Datenbanksystemen zur Verfügung stehen, dürften nicht benutzt werden.

[1] Plain Old Documentation

Letztendlich entschieden wir uns für MySQL als Datenbanksystem. Es bot die Vorteile, das es frei und somit ohne Lizenzkosten einsetzbar, sowie im Betrieb sehr ressourcenschonend und schnell war. Einschränkungen im Befehlssatz (fehlende Möglichkeit von Subqueries) waren egal da VAMOS, wie erwähnt, nur mit einfachen Befehlen arbeiten sollte.

1.3 Pfade von VAMOS am DESY-Zeuthen

VAMOS liegt zur Zeit im AFS unter /project/VAMOS dort liegen auch noch alte Dinge rum, die zur Zeit genutzten Verzeichnisse sind:

prod	Produktionsumgebung (vamos)
test	Testumgebung (tvamos)
devel	Entwicklungsumgebung (dvamos)

Zum Ausführen von Skripten in diesen Verzeichnissen muß die $VAMOS Umgebungsvariable gesetzt sein. Dies geht am einfachsten mit *ini <Wert in der Klammer>*. Nun enthält $VAMOS den Pfad zur aktuellen Umgebung.

2 Die Datenbankstruktur

2.1 Forderungen

VAMOS sollte die Beziehungen der Daten untereinander zu einem großen Teil von sich aus herstellen können. Dadurch wäre es möglich, jegliche relationale Datenbank ohne große vorherige Konfiguration zu benutzen. Aus diesem Grund müssen VAMOS-taugliche Datenbanken einige Bedingungen erfüllen:

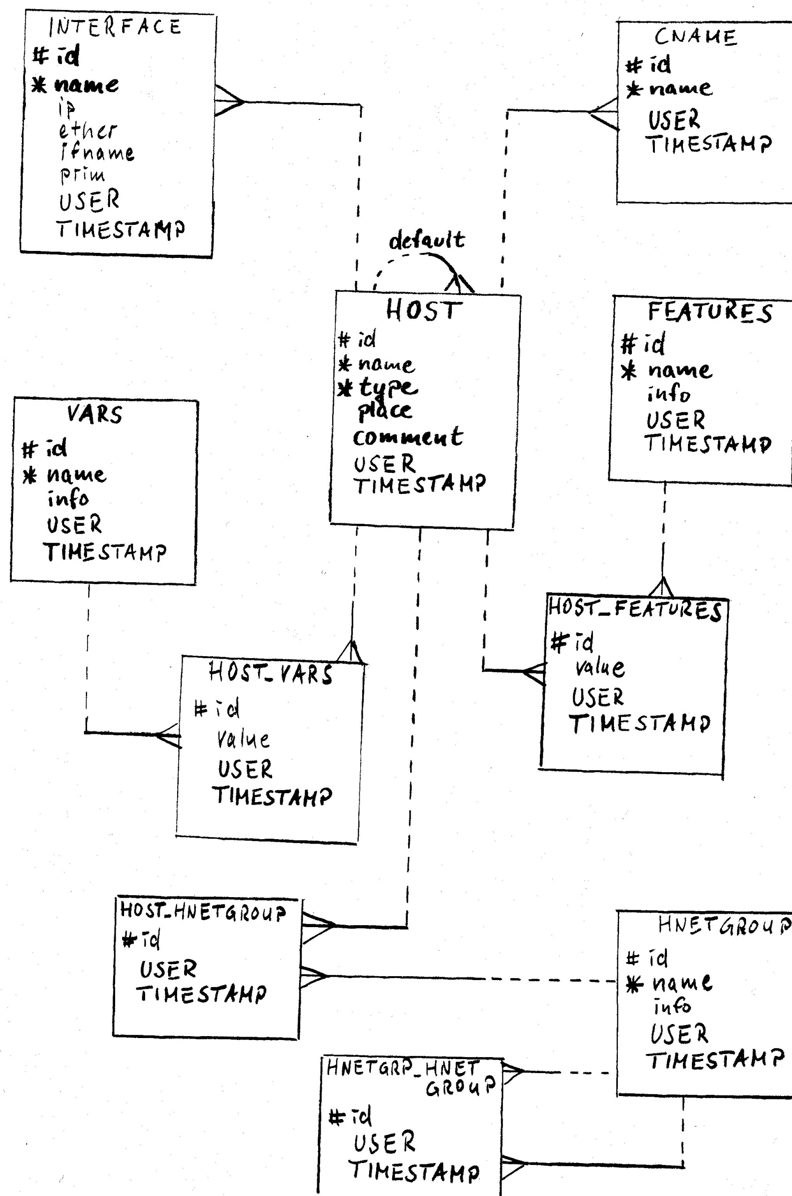
- Der Primärschlüssel heißt immer „id“
- Fremdschlüssel enden immer auf „_id“. Davor sollte der Name der Tabelle stehen, auf die der Fremdschlüssel zeigt. Ist das nicht möglich, muß das VAMOS in der Konfiguration mitgeteilt werden.
- Koppeltabellen, die m:n Beziehungen zwischen Tabellen auflösen, sollten folgende Namensgebung bekommen: „<Name Tabelle 1>_<Name Tabelle 2>“. Ist das aus einem Grund nicht möglich, muß VAMOS das mitgeteilt bekommen.

- Spalten, die hierarchische Self-Joins implementieren, sollten immer „default_id“ heißen.
- Alle Tabellen, die nicht zur Auflösung von m:n-Beziehungen dienen, sollten eine Bezeichnungsspalte haben, die leicht für Menschen lesbar ist. Bestenfalls heißt sie „name“. Sind die Werte dieser Spalte allein nicht eindeutig unter allen Datensätzen (unique-constraint), müssen mehrere Spalten dazu zusammengefaßt werden.

Bereits GenuAdmin unterstützte die Vererbung von bestimmten Eigenschaften bei Objekten. Auch VAMOS sollte dies implementieren können. Zu diesem Zweck muß in jede Tabelle, deren Datensätze auch Vererbung nutzen, eine Spalte „default_id“ existieren. Die Wurzel(n) der Hierarchie besitz(t/en) als „default_id“ NULL.

2.2 Struktur der VAMOS Datenbank

Folgendes ER-Diagramm soll die zur Zeit vorhandene Datenbank erklären (veraltet - vielleicht macht ja mal jemand ein neues).



2.3 Erklärung der Datenbankstruktur

In der Tabelle „host“ sind alle Geräte eingetragen, die am LAN hängen. Dazu gehören PCs, Server, Notebooks, Drucker, Router, Switches, Hubs und ähnliche Devices. Da diese Tabelle hierarchisch aufgebaut ist, existieren hier auch Defaults. Dies sind keine realen Geräte, sie besitzen nur bestimmte Eigenschaften, die andere Defaults oder echte Geräte erben können.

Alle realen Geräte besitzen eine oder mehrere Netzwerkkarten, mit denen sie an das LAN angeschlossen sind. Jede Karte bildet einen Datensätz der Tabelle „interface“. Dort sind ihre Eigenschaften festgehalten. Die Tabelle besitzt eine Spalte „host_id“. Dieser Fremdschlüssel zeigt auf das Gerät (der Tabelle host) in dem die Karte eingebaut ist.

Bestimmte „hosts“ können Aliasnamen haben, wie z.B. „www“ oder „ftp“. Diese sind in der Tabelle „cname“ gespeichert. Da jeder Alias nur auf einem host gesetzt sein darf, ist dies eine 1:n-Beziehung.

Geräte aus der Tabelle „host“ können verschiedene Dienste zur Verfügung stellen. Alle Dienste zusammen sind in der Tabelle „features“ aufgeführt. Da mehrere Geräte den selben Dienst zur Verfügung stellen können, aber auch ein Gerät mehrere Dienste, ist dies eine m:n-Beziehung. Über die Koppeltabelle „host_features“ können die einzelnen Dienste auf den verschiedenen Geräte auf folgende Werte gestellt werden:

prod:	produktiver Einsatz
test:	testweise Einsatz
no:	Dienst abgeschaltet (wie nicht setzen des Dienstes)
noautoprod:	produktiver Einsatz aber kein automatischer Start
noautotest:	testweise Einsatz aber kein automatischer Start

Diese Informationen sind vererbbar. Wenn der default (oder einer seiner defaults) eines „host“-Datensatzes einen Dienst auf einen dieser Werte bereits gesetzt hat, so ist dies automatisch auf diesen Datensatz vererbt worden. Nur wenn er hier anders gesetzt werden soll, muß es überschrieben werden.

Die Geräte der Tabelle „host“ haben auch verschiedene Eigenschaften. Alle Eigenschaften sind in der Tabelle „vars“ aufgeführt. Wieder existiert die Koppeltabelle „host_vars“ mit der diese Eigenschaften auf den Geräten gesetzt werden können, da auch dies eine m:n Beziehung ist. Auch diese Informationen sind auf die gleiche Weise vererbbar wie bei „host_features“.

Um zu regeln, welcher Nutzer sich auf welchen Maschinen anmelden darf, wird in diesem LAN auf Netzgruppen zurückgegriffen. Alle Nutzer sind in bestimmten Nutzernetzgruppen eingetragen, die Maschinen in Hostnetzgruppen. Diese Hostnetzgruppen sind

in der Tabelle „hnetgroup“ aufgeführt. Über die Koppeltabelle „host_hnetgroup“ können Rechner zu bestimmten Hostnetzgruppen hinzugefügt werden. Auch dies ist wieder expandierbar, so daß Netzgruppen im Default auch für den erbenden „host“ gelten. Da es auch die Möglichkeit gibt, daß Hostnetzgruppen in anderen Hostnetzgruppen existieren, kann dies über Einträge in der Koppeltabelle „hnetgrp_hnetgroup“ realisiert werden.

Außerdem fällt auf, daß in jeder Tabelle zwei Felder definiert sind: USER und TIMESTAMP. Diese speichern ab, welcher Nutzer zu welcher Zeit einen Datensatz zuletzt geändert hat. TIMESTAMP wird von MySQL selbständig aktualisiert. USER muß von VAMOS bei jeder Schreiboperation auf die Datenbank mit gesichert werden.

3 Die Repräsentation der Daten im Speicher

3.1 Überlegungen

Um die Datenbankstruktur in eine leicht verarbeitbare und vor allen Dingen lesbare Form zu bringen, waren einige Überlegungen notwendig. Unter Perl können Datenbeziehungen sehr einfach in Form von Hashtabellen (oder assoziativen Arrays) aufgebaut werden. Es entsteht dabei eine Verzeichnisstruktur, wie z.B. auf Dateisystemen. Dadurch, daß Wörter anstatt Zahlen als Index für die „Verzeichnisse“ dienen, ist es für Menschen sehr leicht lesbar. Diese Form der Datenverwaltung sollte auch von den Objekten genutzt werden, die VAMOS erstellt.

Datensätze und ihre Relationen können immer von zwei verschiedenen Seiten betrachtet werden. Als Beispiel kann man sich die m:n Beziehung zwischen Personen und Accounts auf Computern. Eine Person kann mehrere Accounts besitzen. Gleichzeitig ist es aber auch möglich, daß ein (Service-) Account von mehreren Personen genutzt wird. Es besteht also immer das Problem von welcher Seite die Daten aus betrachtet werden. Diese Schwierigkeit hat man immer beim Übertragen einer relationalen Struktur in eine objektorientierte. Wird das Verhältnis von Seiten der Person angesehen, werden seine Accounts zu Unterobjekten der Person. Andersherum gesehen werden Personen zu Unterobjekten des Accounts. Beide Ansichten ergeben einen Sinn bei der Datenbetrachtung. Will man sehen, welche Accounts eine Person besitzt oder lieber wissen, wer alles einen bestimmten Account benutzt. Den Datensatz, aus dessen Position die Datenbeziehungen betrachtet werden, bezeichneten wir als Top-Objekt. Alle Relationen zum Top-Objekt wurden diesem untergeordnet und hießen Unterobjekte.

Ein weiteres Problem trat auf: Auch Unterobjekte konnten wiederum Relationen zu anderen Datensätzen besitzen. Sollten diese mit in die Objektstruktur aufgenommen werden? Wir entschlossen uns für nein! Sollten diese Daten für den Betrachter von Interesse

sein, konnte er jederzeit das Unterobjekt zu einem Top-Objekt machen und sah die Relation aus dieser Sicht. Somit wurde festgelegt, daß die Verschachtelungstiefe der Relationen immer 1 betrug. Dies brachte auch Vorteile (Vereinfachungen) in der Arbeit mit der Datenstruktur mit sich. Das wird sich später noch zeigen.

3.2 Beispiel

```
'cname' => HASH(0x8a755c8)
   empty hash
'features' => HASH(0x8a756d0)
  'juno' => HASH(0x8a75724)
    'xf86' => 'test'
'hnetgroup' => HASH(0x8a73d4c)
  'juno' => HASH(0x8a73db8)
    'lindesk-rz' => 'lindesk-rz'
    'linux-dhcpserver' => 'linux-dhcpserver'
'host' => HASH(0x8a73da0)
  'juno' => HASH(0x8a75574)
    'comment' => '# Linux-Desktop-PC'
    'default' => 'lindesk4-rz-def'
    'name' => 'juno'
    'type' => 'host'
'interface' => HASH(0x8a75520)
  'juno' => HASH(0x8a7561c)
    'ether' => '00:e0:18:98:c0:e4'
    'ifname' => 'eth0'
    'ip' => '141.34.2.230'
    'name' => 'juno'
    'prim' => 1
    'type' => 'ether'
'vars' => HASH(0x8a7573c)
  'juno' => HASH(0x8a757a8)
    'CF_AFSSOFTWARE' => 'openafs-1.2.2'
    'CF_USER' => 'ahaupt'
    'Linux_kernel_version' => '2.4.16-z4 2.4.7-z1'
```

```
'cname' => HASH(0x8ebc1a4)
   empty hash
'features' => HASH(0x8ebd970)
  'juno' => HASH(0x8ebe430)
    'xf86' => 'test'
  'lindesk4-rz-def' => HASH(0x8ebe16c)
    'ldap' => 'test'
  'linux-4-def' => HASH(0x8ec10c8)
    'afs_client' => 'test'
    'afs_software' => 'test'
    'arc' => 'no'
    'ypserv' => 'test'
  'linux-4-desktop-def' => HASH(0x8ebe10c)
    'cdrom' => 'test'
    'hepixdm' => 'test'
  'unix-def' => HASH(0x8ebe3f4)
    'CellServDB' => 'no'
    'skey' => 'no'
    'wu-ftp' => 'no'
'hnetgroup' => HASH(0x8a7893c)
  'juno' => HASH(0x8a7b0bc)
    'lindesk-rz' => 'lindesk-rz'
    'linux-dhcpserver' => 'linux-dhcpserver'
  'linux-4-desktop-def' => HASH(0x8a7b098)
    'linux-desktop' => 'linux-desktop'
'host' => HASH(0x8a7af18)
  'juno' => HASH(0x8ebc150)
    'comment' => '# Linux-Desktop-PC'
    'default' => 'lindesk4-rz-def'
    'name' => 'juno'
    'type' => 'host'
'interface' => HASH(0x8ebc0fc)
  'juno' => HASH(0x8ebc1f8)
    'ether' => '00:e0:18:98:c0:e4'
    'ifname' => 'eth0'
    'ip' => '141.34.2.230'
    'name' => 'juno'
    'prim' => 1
    'type' => 'ether'
'vars' => HASH(0x8ebda3c)
  'ifh-def' => HASH(0x8ec4614)
    'mx' => '10 znsun1.ifh.de.'
  'ifh-domain-def' => HASH(0x8ec456c)
    'domain' => 'ifh.de'
    'ypdomain' => 'zhepnet'
  'juno' => HASH(0x8ec6258)
    'CF_AFSSOFTWARE' => 'openafs-1.2.2'
    'CF_USER' => 'ahaupt'
    'Linux_kernel_version' => '2.4.16-z4 2.4.7-z1'
  'lindesk4-rz-def' => HASH(0x8ec4644)
    'CF_PASSWD' => 'lindesk-rz'
    'CF_RPM_ADD_AGE' => 1
    'CF_YPSERVER' => 'ajax'
  'linux-4-def' => HASH(0x8ec55c0)
    'AFS_SYSNAME' => 'i386_linux24'
    'RunNetWorker' => 'false'
  'linux-4-desktop-def' => HASH(0x8ec42b4)
    'CF_CDROM' => 'hdc'
    'CF_CONSOLE' => 'true'
  'unix-def' => HASH(0x8ec5614)
    'AUTOMOUNT' => 'amd'
    'utils' => 'utils'
```

3.3 Erklärung der Datenstruktur

Auf der linken Seite sind die Daten eines PCs zu bewundern, auf der rechten auch, nur ist er hier „expandiert“. Das heißt um die Daten seiner Defaults erweitert. Man sieht, daß die Daten in Sektionen und Untersektionen bis zur dritten Ebene unterteilt sind.

Die erste Ebene korrespondiert mit der Tabelle aus der die Daten geholt werden. Es ist der eigentliche Name der Tabelle oder wenn es sich um eine Koppeltabelle handelt, um den Namen der Top-Objekt-Tabelle gekürzt. Beispiel: Aus der Tabelle „host_features“ wurde der Eintrag „features“. Die zweite Ebene zeigt auf, von welchem Top-Objekt (in diesem Fall Host) die Daten kommen. Dies ist natürlich nur beim expandieren sinnvoll. Aber um einen einheitlichen Zugriff zu gewähren, ist es immer so. Der Eintrag „interface“ bildet eine Ausnahme. Hier ist die zweite Ebene nach dem Namen des Interfaces benannt, es können auch mehrere existieren (Vergleich 4.4). Die dritte Ebene enthält die eigentlichen Daten.

4 Das Perl-Interface zur Datenbank

4.1 Übersicht

Aufgrund der Komplexität von VAMOS wurden die entwickelten Module und Konfigurationsdateien im VAMOS-Verzeichnis in folgende Unterverzeichnisse abgelegt.

conf:	Konfigurationsdateien
data:	zwingend benötigte Basismodule
gui:	VAMOS-GUI Module
server:	Module des VAMOS Proxyservers
client:	Client-Module zum Nutzen des VAMOS Proxyservers
scripts:	Skripte, die die VAMOS-Module nutzen
doc:	Dokumentationsverzeichnis

Wie bereits erwähnt, war es auch meine Aufgabe, das Perl-Interface zur Datenbank zu entwickeln. Dieser Teil von VAMOS wurde aufgrund der Komplexität in mehrere Module aufgeteilt (Unterverzeichnis data):

- Db.pm: – Die Datenbankschnittstelle
- Top_Object.pm: – Zugriff auf Top-Objekte
- Tables.pm: – Funktionalität der Unterobjekte
- Include_Objects.pm: – Definition der Top_Object- und Unterobjektklassen
- SynCheck.pm: – Syntaxchecking der Eingaben
- VamosLog: – Funktionen für Logging / Fehlerbehandlung
- ConfigVars.pm: – Zugriff auf Konfigurationsdateien
- Possible_Values.pm – Vorgabe von möglichen Werten bestimmter Attribute

4.2 Db.pm

Dieses Modul ist die Schnittstelle zwischen VAMOS und der Datenbank. Generell sind keine Einschränkungen bezüglich eines bestimmten Datenbanksystem vorgesehen. VAMOS benutzt nur einfache Standard-SQL Befehle. Aus diesem Grund sollte jedes SQL-fähige System funktionieren, für das es einen Perl DBD[2]-Treiber gibt. Für diese Implementierung wurde MySQL aus den bereits erwähnten Gründen gewählt.

Die Angaben, welche Datenbank auf welchem Host eines bestimmten Datenbanksystems genutzt werden soll, werden in der Konfigurationsdatei vamos.conf in der Sektion DATABASE eingetragen.

Kernstück des Moduls ist die Funktion Init_DB. Sie verbindet sich mit der Datenbank und liest sie in den Hauptspeicher in Form von Hashtabellen komplett ein. Dies ist zur Zeit nötig und funktioniert bei kleinen bis mittleren Datenbanken auch problemlos. Für große Datenbanken ist diese fehlende Skalierbarkeit natürlich der Knock Out, wenn nicht genügend RAM zur Verfügung steht. Sollen auch solche riesigen Datenmengen bearbeitbar sein, müßte an diesem Konzept noch etwas gefeilt werden.

Wie gesagt werden die einzelnen Datensätze in Hashtabellen geladen. Sie heißen allgemein „<Name der Tabelle>_hash“. Eine Tabelle mit dem Namen „host“ würde also in einer Hashtabelle mit dem Namen „host_hash“ abgespeichert. Als Index wird der Wert der Spalte id genutzt. Dieser Eintrag enthält nun eine Referenz auf eine Hashtabelle, in der die Spaltennamen die Indizes sind, die auf Werte des Datensatzes zeigen.

Das ist aber nur der erste Schritt. Im zweiten Schritt werden alle Fremdschlüssel als Index wiederum in Hashtabellen gespeichert. Dies beschleunigt die spätere Verknüpfung der Datensätze enorm. Der Name der Hashtabelle wird wie folgt generiert: <Name der Tabelle>_<Name des Fremdschlüssels>_hash. Als Index des Hashes werden die Fremdschlüssel genommen. Der Wert zeigt auf die kompletten Datensätze, die wie im oberen Absatz beschrieben schon erstellt wurden. Ein Beispiel soll das verdeutlichen: In der Tabelle

[2] Database Dependent

„interface“ hat ein Datensatz den Primärschlüssel id=212 und als Fremdschlüssel „host_id“ mit dem Wert „123“. Man kann über „interface_host_id_hash{123}{212}“ auf diesen Datensatz zugreifen. Wenn man sich alle Schlüssel des „interface_host_id_hash{123}“ ausgeben läßt, weiß man sofort, welche interfaces am host mit der id 123 hängen (hier wäre das interface mit der id 212 darunter). Dies läßt das Zusammensetzen der Objekte (wie in den nächsten Punkten beschrieben) rasant von statten gehen, da die zusammen gehörigen Datensätze nicht erst gesucht werden müssen.

In diesem Modul befinden sich außerdem die Funktionen zum Generieren und Absetzen der SQL Statements. Diese werden durch Auswertung einer Datenstruktur (wird unter 4.4 erklärt) erzeugt. Die Hashtabellen, die in den letzten Abschnitten erklärt wurden, werden bei der Änderung der Daten auf der Datenbank auch im Speicher angepaßt. Dies deshalb, weil die geänderten Objekte (aus 4.4) aus den Hashes danach neu aufgebaut werden.

4.3 TopObject.pm

Dieses Modul stellt eine abstrakte Klasse zur Arbeit mit Top-Objekten zur Verfügung. Abstrakt deshalb, weil nur aus von ihr vererbten Klassen Objekte gebildet werden können. Die abgeleitete Klasse muß exakt den selben Namen haben, wie die Tabelle aus deren Datensätzen Top-Objekte gebildet werden sollen.

Die Klasse „top“, die in diesem Modul implementiert ist, arbeitet als Abstraktionsschicht zwischen dem Programm und den Unterobjekten (in 4.4 erklärt), die die eigentlichen Daten enthalten. Sie ermöglicht einen einheitlichen Zugriff und faßt zusammengehörige Daten zum einem einzigen Objekt zusammen. Die elementaren Datenbankoperationen Lesen, Anlegen, Ändern und Löschen werden von hier aus gesteuert.

Alle Objekte werden in einem Hash (mit ihrem Namen als Schlüssel) vorgehalten, so daß sie bei Anforderung nicht immer neu erstellt werden müssen.

4.4 Tables.pm

Dieses Modul enthält eine Reihe von abstrakten Klassen, die verschiedene Relationen aus Datenbanksicht in Perlobjekte umsetzen. In Zusammenarbeit mit dem Modul Top_Object entsteht so die Datenstruktur, die unter 3.2 vorgestellt wurde.

Zur Zeit existieren folgende Klassen (Beziehungen aus Sicht des Top-Objekts):

table	– Basisklasse für jede weitere Klasse
norm_table	– „normale“ 1:n-Beziehung
list_table	– 1:n-Beziehung, nur Auflisten der Werte einer Spalte
top_table	– Tabelle, aus der das Top-Objekt kommt
interface_table	– Auflösung m:n Beziehung mit genau einem Wert
norm_interface_table	– Auflösung m:n Beziehung mit mehr als einem Wert
list_interface_table	– m:n-Beziehung, nur Auflistung wie list_table

Auch in diesem Modul sind die Klassen wieder abstrakt. Das heißt, es müssen weitere Klassen definiert werden, die von der hier gegebenen Auswahl erben. Im allgemeinen muß dabei folgende Namensgebung eingehalten werden (im Zusammenhang mit den Punkten aus 2.1): Klassenname = <Name der Top-Objekt Tabelle>_<Name der Tabelle, die implementiert werden soll>. Dies gilt für norm_table, list_table, top_table. Für (list_)interface_table gilt folgender Grundsatz: Klassenname = <Name der Top-Objekt Tabelle>_<Name der Tabelle zu der m:n Beziehung aufgelöst werden soll>.

Dies ist wie gesagt der Standardfall. Sollte aufgrund der Namensgebung der Tabellen in der Datenbank dieses Konzept nicht eingehalten werden können, muß dies in einem Hash namens „CONVERT_NAMES“ verankert werden. Nur so kann die Klasse „herausfinden“, welche Tabelle sie implementieren soll.

Die Klasse „table“ ist wie schon erwähnt die Basisklasse für alle noch existierenden Klassen in diesem Modul. Sie stellt die Basisfunktionalität zur Verfügung, die jede abgeleitete Klasse besitzen muß. Zu den vier grundlegenden Datenbankoperationen auf einen Datensatz sind hier die passenden Funktionen dazu implementiert:

Auslesen	– Methode Get()
Hinzufügen	– Klassenfunktion Add()
Ändern	– Methode Update()
Löschen	– Methode Delete()

Da jede Klasse (intern) die Daten anders verwaltet und die genannten Methoden nichts darüber wissen können, rufen sie von sich aus an bestimmten Stellen (wo die Implementierung sich zwischen den Klassen unterscheidet) Methoden auf, die in den Unterklassen überschrieben werden müssen. Dort werden die Sonderfälle abgearbeitet und liefern einen fest definierten Wert zurück. Nur in norm_table weicht die Implementierung so stark ab, daß die Methoden Add und Delete dort komplett überschrieben werden mußten.

Die Funktionen Add, Update und Delete generieren bei ihrem Aufruf eine Datenstruktur. In ihr ist die zu erledigende Aktion, die Tabelle und der zu ändernde Datensatz

zusammen mit den neuen Daten enthalten. Liefern sie diese Struktur nicht zurück, ist ein Fehler (Syntaxcheck, ...) aufgetreten. Diese Datenstruktur nutzt das Db-Modul, wie erwähnt, um seine eigene Datenstruktur zu aktualisieren, sowie die SQL-Statements zu generieren. Auch der VAMOS Server nutzt sie (siehe 5.1.1).

Zusammen mit dem Wissen der Datenbankstruktur (aus 2.2) kann nun die Entstehung der Datenstruktur im Speicher (aus 3.2) erklärt werden. Die erste Ebene basiert bekanntlich auf dem Namen der Tabelle. Den Tabellen wurden aufgrund der Beziehungen untereinander folgende Klassen zugeordnet:

host: erbt von „top_table“, da es die Daten des Top-Objekts aufnimmt.

interface: erbt von „norm_table“, da ein Host mehrere Interfaces haben kann, ein Interface muß einem Host zugeordnet werden.

cname: erbt von „list_table“, da nur Auflistung der Aliasnamen entscheidend ist.

hnetgroup: erbt von „list_interface_table“, da m:n Beziehung über Tabelle „host_hnetgroup“ und nur Auflistung der Namen entscheidend.

vars: erbt von „interface_table“, da m:n Beziehung, der Name der Variablen zeigt auf den Wert, Daten genommen aus „host_vars“.

features: wie „vars“, Daten aus „host_features“.

Das Expandieren der Daten erfolgt durch Abfrage der Daten des Defaults und setzen dieser Daten im eigenen Objekt.

4.5 Include_Object.pm

Hier werden die Top_Objekt- und Unterobjektklassen den virtuellen Basisklassen durch Vererbung zugeordnet. Diese sind, wie erwähnt, in Top_Object.pm und Tables.pm definiert. Ein Anwendungsprogramm muß nur diese diese Klasse einbinden, um die Funktionalität zu nutzen.

4.6 SynCheck.pm

Dieses Modul stellt zur Zeit nur eine Funktion namens „Check“ zur Verfügung. Sie überprüft mit Hilfe der Konfigurationsdatei $VAMOS/conf/synchk.cfg und dem Modul Possible_Values.pm (4.9) ob ein bestimmter Wert in die Spalte eines Datensatzes paßt. Die

Funktion verlangt zwei Parameter: eine Hashstruktur, die die durchzuführende Änderung beschreibt und ein Datenbankhandle, mit dem bestimmte Werte in der Datenbank überprüft werden. Als Rückgabewert liefert die sie einen „true“-Wert zurück, wenn alle Tests erfolgreich durchlaufen wurden und der Wert eingetragen werden kann, andernfalls „false“.

Als erstes wird untersucht, ob der Wert den Typanforderungen der Datenbank gerecht wird. Im zweiten Schritt wird die Konfigurationsdatei (synchk.cfg) nach der Tabelle-Spalte-Kombination abgesucht. Dort können reguläre Ausdrücke oder kleine Perlfunktionen eingetragen werden. Diese Prüfung muß der übergebene Wert überstehen (besitzt eine IP-Adresse das richtige Format?, ...). Im dritten Schritt ist das Eindeutigkeitschecking an der Reihe. Ist diese Spalte in der Konfigurationsdatei mit einem Unique-constraint versehen, wird letztendlich noch untersucht, ob schon ein Datensatz mit diesem Wert in der Tabelle existiert.

4.7 VamosLog.pm

Dieses Modul stellt die Funktionalität zur Verfügung, um Log- und Debuginformationen, Warnungen und Fehlermeldungen zu sammeln. Es ist die zentrale Schnittstelle für Meldungen dieser Art im VAMOS System. Die Ausgabe erfolgt standardmäßig auf STDERR. Dies geschieht durch eine interne Funktion, die aber von Programmen überschrieben werden kann. Auf diese Weise können die Meldungen überall hin ausgegeben werden.

4.8 ConfigVars.pm

Dieses Modul sorgt dafür das die globale Konfiguration von VAMOS in den Speicher geladen wird. Dies geschieht beim Start des Programms. Einstellungen können von den verschiedenen Modulen über Variablen im ConfigVars Modul abgefragt werden. Es dient zur Beschleunigung des Programmablaufs. Auf diese Weise muß nicht jedes Modul, daß eine Einstellung auslesen will, selbst auf die Konfigurationsdatei zugreifen.

4.9 Possible_Values.pm

Dieses Modul stellt für einige Attribute in bestimmten Tabellen mögliche Werte zur Verfügung. Diese werden teilweise auch von SynCheck.pm (4.6) zur Verifikation von Eingaben genutzt.

Die Vorgabewerte werden mit drei verschiedenen Methoden ermittelt:

1. Das Attribut ist in der Datenbank ein Fremdschlüssel: Gültig sind alle Namen von Objekten in der Tabelle, auf die der Fremdschlüssel zeigt. Hierzu können auch Einschränkungen in $VAMOS/conf/vamos.conf in der Sektion [VALUE_RESTRICTIONS] definiert werden.

2. Das Attribut ist in der Datenbank eine Menge: Alle Werte aus dieser Menge können genommen werden. Änderungen an diesen Werten können jedoch nur direkt in der Datenbank vorgenommen werden und erfordern einen Neustart des VAMOS-Servers. Als Beispiel soll ein zusätzlicher Typ „test“ für Netwerkkarten in der Tabelle „interface“ eingeführt werden:

```
mysql> desc interface;
+---------+---------------------+------+-----+---------+----------------+
| Field   | Type                | Null | Key | Default | Extra          |
+---------+---------------------+------+-----+---------+----------------+
| id      | int(10) unsigned    |      | PRI | NULL    | auto_increment |
| host_id | int(10) unsigned    | YES  | MUL | NULL    |                |
| ether   | varchar(17)         | YES  | MUL | NULL    |                |
| ip      | varchar(17)         | YES  | MUL | NULL    |                |
| name    | varchar(32)         |      | UNI |         |                |
| ifname  | varchar(8)          | YES  |     | NULL    |                |
| type    | set('ether','fddi') | YES  |     | ether   |                |
| prim    | set('0','1')        | YES  |     | 0       |                |
| USER    | varchar(32)         | YES  |     | NULL    |                |
| TIME    | timestamp(14)       | YES  |     | NULL    |                |
+---------+---------------------+------+-----+---------+----------------+
10 rows in set (0.00 sec)

mysql> alter table interface
    -> modify type set('ether','fddi','test') default 'ether';
Query OK, 1310 rows affected (0.55 sec)
Records: 1310  Duplicates: 0  Warnings: 0

mysql> desc interface;
+---------+----------------------------+------+-----+---------+----------------+
| Field   | Type                       | Null | Key | Default | Extra          |
+---------+----------------------------+------+-----+---------+----------------+
| id      | int(10) unsigned           |      | PRI | NULL    | auto_increment |
| host_id | int(10) unsigned           | YES  | MUL | NULL    |                |
| ether   | varchar(17)                | YES  | MUL | NULL    |                |
| ip      | varchar(17)                | YES  | MUL | NULL    |                |
| name    | varchar(32)                |      | UNI |         |                |
| ifname  | varchar(8)                 | YES  |     | NULL    |                |
| type    | set('ether','fddi','test') | YES  |     | ether   |                |
| prim    | set('0','1')               | YES  |     | 0       |                |
| USER    | varchar(32)                | YES  |     | NULL    |                |
| TIME    | timestamp(14)              | YES  |     | NULL    |                |
+---------+----------------------------+------+-----+---------+----------------+
10 rows in set (0.00 sec)

mysql>
```

3. Das Attribut ist ein Wert in einer Koppeltabelle (z.B. host_vars) und es existiert eine Tabelle <Tabelle1>_values_<Tabelle2>: Die Werte kommen aus der weiteren Tabelle (in diesem Fall vars_values_host), in der die Werte gespeichert sind. Da die

Tabelle mit vars beginnt, sind die Werte über ein vars-Topobjekt editierbar (begänne sie mit host_ wäre es über ein host-Objekt möglich).

5 Die Client / Server Architektur

Eine weitere Forderung an das System war es, vom Konzept her, die Zugriffsrechte auf Datensätze beschränken zu können. Einzelne Aktionen sollten für bestimmte Nutzergruppen möglich sein, andere nicht. Dies sollte nicht sofort mit implementiert werden, da VAMOS zunächst nur als ein Administrationswerkzeug konzipiert war.

Access Control Lists oder ähnliche Zugriffssteuerungsmethoden werden von den verschiedenen Datenbanksystemen aber nur unzureichend implementiert. So kann MySQL nur Zugriffsrechte bis auf Spaltenebene in Tabellen umsetzen. Die erste Realisierung der Zugriffssteuerung wurde somit schließlich verworfen.

Diese erste Realisierung war folgendermaßen konzipiert. Das Nutzerprogramm erstellt ein Kerberos-Ticket des Nutzers, der gerade damit arbeitet. Dieses wird zu einem Server geschickt. Dieser entschlüsselt das Kerberos-Ticket und sieht nach, ob der entsprechende Nutzer in einer Liste aufgeführt ist. Wenn ja, wird der Datenbanknutzername sowie das dazu gehörige Paßwort zurückgeschickt, andernfalls nicht. Nur mit Nutzernamen und Paßwort kommt der Benutzer an die Daten. Verschiedene Benutzer hätten verschiedene Nutzernamen-Paßwort-Kombinationen bekommen können, die Zugriffssteuerung wäre vom Datenbanksystem übernommen worden. Aufgrund der Mängel der verwendeten Datenbank mußte dieses Konzept allerdings überarbeitet werden.

Im Gegensatz dazu sollte nun der Server selbst diese Zugriffssteuerung übernehmen. Um dies umzusetzen, mußte das ganze VAMOS-System zentralisiert werden. Der Server gibt nun nicht mehr den Zugriffcode zurück, sondern hält selbst die Daten vor und stellt sie dem Clienten auf Anfrage zur Verfügung. Ob er das wirklich tut oder seinen Dienst wegen unzureichender Rechte verweigert, wird anhand von ACLs festgelegt. Dies ist zur Zeit so implementiert, daß es funktioniert – vom Design ist es allerdings noch nicht berauschend.

Grundlage der Funktionalität ist das CPAN-Modul PlRPC. Die darin enthaltenen Klassen können zur Implementierung von Remote Procedure Calls genutzt werden. Meine Klassen VamosServer und VamosClient erben von diesem genialen Modul.

5.1 Der VAMOS-Server

Alle Module, die unter 4.1 erwähnt wurden, kommen nun auf dem Server selbst und nicht mehr auf der Clientseite beim Benutzer zum Einsatz. So muß nun beim Start des Ser-

vers die komplette Datenbank in den Hauptspeicher geladen werden. Auch die einzelnen Objekte werden hier erstellt. Der Client fordert sie nur an.

5.1.1 Allgemeine Probleme

VAMOS ist ein Mehrbenutzersystem. Das heißt, der Server muß mehrere Anfragen von unterschiedlichen Clienten bedienen können. Aus diesem Grund ist er als forkender Server konzipiert. Jeder Client erhält eine Verbindung mit genau einem Serverprozeß. Dies bringt einige zusätzliche Probleme mit sich, die ich, zusammen mit der Lösung, aufzeigen werde.

Um auf die Daten der Datenbank zuzugreifen nutzt der Server die vererbten Klassen des in 4.3 vorgestellten Top_Object Moduls. Dies geschieht über die AUTOLOAD-Funktion von Perl. Von diesem Konzept mache ich in mehreren Modulen Gebrauch. Ich werde sie hier kurz vorstellen: Wenn der Perl-Interpreter im aktuellen Namensraum beim Funktionsaufruf keine passende Funktion mit diesem Namen findet, wird zuletzt (vor einem Abbruch) die AUTOLOAD Funktion aufgerufen. Als Parameter bekommt sie den Namen der eigentlich aufzurufenden Funktion übergeben. Als ein weiterer Parameter wird der Klassenname mit übergeben, in der sich die Funktion befindet. Dies geschieht durch einen Mechanismus, den ich später noch beschreibe. Nun ruft die AUTOLOAD Funktion selbst die eigentlich zu rufende Funktion auf (mit den weiteren Parametern, die sie noch bekommen hat) und gibt deren Rückgabewert zurück. Wenn die aufzurufenden Klassen auf der Client- und der Serverseite gleich heißen, bekommt man so einen völlig transparenten Proxyserver. Das aufrufende Programm „merkt“ davon nichts. Es kann als Client oder auch als „normale“ Applikation ausgeführt werden, bei der alle Daten lokal im Hauptspeicher liegen.

Aus den zwei Konfigurationsdateien vamos.conf und server.conf erhält der Server die Logininformationen für die Datenbank (Rechnername, Datenbankname, Datenbanksystem, Datenbanknutzer und Paßwort). In vamos.conf stehen die allgemein zugänglichen Daten. Server.conf darf nur der Server selbst lesen, da hier die sensiblen Informationen enthalten sind (Datenbanknutzer, Paßwort). Außerdem ist hier die Liste der VAMOS-Administratoren eingetragen.

Ein Problem, daß es zu bewältigen galt, war die Übertragung der Logginginformationen vom Server auf den Client. Diese können nicht einfach geschickt werden. Der Server erbt von einer Klasse, die das nicht unterstützt. Der Client nimmt nur Daten an, wenn er vorher eine Anfrage gesendet hatte. Also werden sie bei jedem Funktionsaufruf, der über den Server geht, bei der Rückgabe quasi „huckepack“ mit übertragen. Alle Meldungen von VAMOS werden außerdem zusammen mit zusätzlichen Informationen in ein Logfile geschrieben.

5.1.2 Konfiguration

Der VAMOS Server besitzt eine eigene Konfigurationsdatei lokal auf dem Serverhost (normalerweise /var/site/vamos/conf/server.conf). Dort sind sicherheitsrelevante Daten gespeichert. Der Rest kommt aus der „normalen“ Konfigurationsdatei $VAMOS/conf/vamos.conf.

Änderungen an diesen beiden Dateien erfordern oft den Neustart des VAMOS Servers, während Änderungen an Zugangsberechtigungen (ACLs, ...) beim nächsten Login ohne Neustart wirksam werden.

5.1.3 Der Loginprozeß

Die Authentisierung mußte ich komplett neu implementieren. Die Klasse RPC::PlServer besitzt zwar selbst ein einfaches Authentisierungssystem (Nutzername / Paßwort), dies war jedoch für diesen Anwendungsfall nicht geeignet, da jeder Nutzer auch noch ein extra Paßwort für VAMOS hätte bekommen müssen - die Verifizierung über z.B. PAM stellt diese Klasse nicht zur Verfügung.

Zur Zeit werden drei verschiedene Authentisierungsmechanismen unterstützt:

Kerberos5: Benötigt das Perl Modul Authen::Krb5. Authentisiert wird über Kerberos5.

Kerberos4: Benötigt das Perl Modul Authen::Krb4. Authentisiert wird über Kerberos4.

RSA: Der Nutzername wird mit Hilfe von Crypt::OpenSSL::RSA signiert und die Signatur auf dem Server verifiziert. Der Nutzer muß vom Admin einen Private-Key bekommen und diesen als $HOME/.vamos/private_key speichern. Dieser darf dort nur für den Nutzer lesbar sein! Der passende Public-Key wird auf dem Server als $VAMOS/sslkeys/rsa_public_key-$username gespeichert. Das Keypaar kann mit Hilfe des Scripts gen_keypair erzeugt werden.

PAM: Authentisiert wird mit der Nutzername / Paßwort Kombination. Auf dem Server muß eine PAM Konfigurationsdatei /etc/pam.d/vamos (Pfad für Linux) liegen. Als Servicename wird nur „auth“ unterstützt – es reicht also ein Eintrag, der gegen eine Paßwortdatenbank authentisiert.

Wird kein bestimmter Mechanismus beim Login angegeben (als Parameter des New-Konstruktors der Klasse VamosClient), werden sie in der Reihenfolge, in der sie in vamos.conf angegeben sind, durchgetestet. Dies entspricht einem SASL-ähnlichen Verhalten. Sobald einer erfolgreich war, ist der Nutzer angemeldet.

Sowohl im „server“ als auch im „client“ Verzeichnis unter $VAMOS liegt ein Modul Auth.pm und ein Verzeichnis Auth, daß die Implementationen der einzelnen Loginmethoden enthält. Auth.pm benutzt die Module über eine einheitliche Schnittstelle, die Auth::Base vorgibt. Die nötigen Daten, um ein Login erfolgreich durchzuführen, werden von den client-Modulen gesammelt. Die Loginfunktion von VamosClient schickt diese an den Server, wo sie von den Authentisierungsmodulen auf der Serverseite ausgewertet werden. Genauere Einzelheiten können den einzelnen Modulen entnommen werden.

5.1.4 Datensynchronisation zwischen den Serverprozessen

Auch das Problem der Wahrung der Konsistenz der Daten zwischen den einzelnen Serverprozessen mußte gelöst werden. Hierzu bediene ich mich der Shared Memory Segmente, die alle SysV-kompatiblen Systeme bieten sollten. Um einen einfachen Zugriff auf solche Segmente zu gewährleisten, existiert ein CPAN-Modul namens IPC::Shareable. Dieser Bereich wird von allen Serverinstanzen genutzt.

Er besteht zur Zeit aus einer Hashreferenz mit folgenden Einträgen:

lock: Nutzername, der zur Zeit einen exclusiven Schreiblock gesetzt hat.

processes: PIDs der aktuell laufenden Prozesse.

clients: Hashstruktur mit Daten der aktuell verbundenen Clients

dataset_locks: Hashstruktur: enthält alle Tabellennamen sowie die id’s der Datensätze als Keys, die gerade in dieser Tabelle gelockt sind – als Wert ist die client_id gespeichert. Dies geschieht zur Zuordnung welche Instanz welche Datensätze gelockt hält.

Sobald ein Nutzer Änderungen an der Datenstruktur macht, stellt die dazugehörige Serverinstanz eine Datenstruktur zusammen, mit der die anderen Instanzen ihre eigene Datenstruktur aktualisieren können. Diese wird in in die Tabelle __Changes der Datenbank gespeichert. Der eigentliche Refresh der Daten im Hauptspeicher muß vom Client angestoßen werden. Entweder ruft er die Servermethode „Refresh“ auf, oder er führt selbst eine Änderung an der Datenstruktur durch. Dabei wird „Refresh“ automatisch aufgerufen. Der Rückgabewert von „Refresh“ ist eine Hashreferenz, die als Keys die verschiedenen Top-Objekt Klassen und als Wert ein Array der Namen, der geänderten Top-Objekte in der jeweiligen Klasse.

5.2 Das Client-Modul

Dieses Modul enthält die Klasse VamosClient, welche die meisten Anfragen an den VAMOS Server durchreicht. Dies geschieht wieder durch die bereits erwähnte Funktion AUTOLOAD. Die Funktionalität erbt VamosClient von der Klasse RPC::PlClient. Die Information auf welchem Rechner und welchem Port der Server „lauscht“, erhält es wieder aus der Datei vamos.conf.

Die Klasse enthält die Funktionen Cipher und Compression. Diese sorgen dafür, daß auf dem Client und dem Server Verschlüsselung und Kompression an- bzw. abgeschaltet werden. Die Funktion Cipher erzeugt zusätzlich beim ersten Aufruf einen zufälligen Sitzungsschlüssel, worauf ich in 5.3.1 noch zurückkommen werde. Weiterhin befindet sich hier die Login-Methode, die automatisch beim Verbinden mit dem Server aufgerufen wird. Dies deswegen, da der Nutzer ohne Anmeldung sowieso keine Daten vom Server anfordern kann.

Eine weitere Aufgabe dieser Klasse ist es, die Logginginformationen, die der Server, wie bereits erwähnt, „huckepack“ mitsendet, wieder herauszufiltern. Dies muß deshalb geschehen, weil die von VamosClient beerbte Klasse mit diesen Daten nicht klar kommt.

5.3 Sicherheit

5.3.1 Verschlüsselung

Um die Daten der Datenbank sowie die Nutzerdaten bei der Anmeldung vor unbefugtem Zugriff zu schützen, kann die Datenübertragung verschlüsselt erfolgen. Die Anmeldung selbst erfolgt immer verschlüsselt.

Zu diesem Zweck kommen sowohl asymmetrische als auch symmetrische Verschlüsselung zum Einsatz. Sie ist dem Konzept, welches auch die SSH nutzt, nachempfunden: Zuerst erzeugt der Client einen zufälligen Sitzungsschlüssel. Dieser wird mit dem Public-Key des VAMOS Servers (RSA-Verschlüsselung von OpenSSL) verschlüsselt. Der kodierte Sitzungsschlüssel wird an den Server übertragen. Er dekodiert ihn mit Hilfe seines Private-Keys. Die weitere Verschlüsselung erfolgt mit Blowfish. Als Passwort kommt der Sitzungsschlüssel zum Einsatz. Aus Performancegründen wird dieser etwas umständliche Weg gewählt, da symmetrische Verschlüsselung bedeutend schneller als ihr asymmetrisches Pendant ist.

5.3.2 Zugriffsschutz / ACLs

Im Client/Server Betrieb findet beim Zugriff auf Objekte ein Check statt, der die Zulässigkeit einer Aktion überprüft. Dieser Check ist modular aufgebaut - d.h. die Klasse, die für

die Prüfung verantwortlich ist, kann ausgetauscht werden. Die Klassen liegen unter $VAMOS/data/ACL. Welche Methoden eine Klasse zur Verfügung stellen muß, kann in den bereits vorhandenen Klassen nachgesehen werden.

Zur Zeit ist die Klasse UserACL aktiv. Das kann in $VAMOS/conf/vamos.conf unter [ACL] jedoch verändert werden. UserACL besitzt verschiedene Aspekte der Zugriffssteuerung. Die Konfigurationsdatei für diese Klasse ist $VAMOS/conf/acl.conf. Eine Beschreibung des Aufbaus ist in der Datei enthalten. Ein VAMOS-Administrator (in der server.conf definiert) darf grundsätzlich alles. Zusätzlich gibt es Tabellenadministratoren, die alle Operationen (bzw. die in acl.conf definierten) auf einer bestimmten vorgegebenen Tabelle ausführen dürfen. Zu guter letzt gibt es noch die Rollen „USER“ und „GROUPADMIN“. Ob ein Nutzer für ein bestimmtes Objekt eine bestimmte Rolle annimmt wird unter den jeweiligen Abschnitten konfiguriert. In den Abschnitten „*-DATASET_MASK“ sind dann besondere Zugriffsrechte für einzelne Attribute des Objektes definierbar, für den der Nutzer diese bestimmte Rolle hat.

5.4 Transparenz beim Datenzugriff

Auch auf der Clientseite gibt es eine abstrakte Top-Objekt Klasse. Auch sie muß wie unter 4.3 beschrieben von weiteren Klassen beerbt werden.

Im Gegensatz zu ihrem Pendant auf der Serverseite, ist diese Klasse völlig trivial. Sie tut nichts weiter, als jeden Aufruf 1:1 an die Klasse VamosClient durch zureichen, welche ihn über RPC an den Server weiterleitet.

So entsteht die bereits beschriebene totale Transparenz. Alle Anwendungen können auf Daten über RPC zugreifen. Auf die selbe Art können sie aber auch, indem sie von der unter 4.3 beschriebenen Top-Objekt-Klasse erben, auf die Daten lokal zugreifen. Dies kann je nach Anwendungsfall entschieden werden.

6 Dateigenerierung

Die Erstellung der Konfigurationsdateien für verschiedene Dienste wird vom VAMOS-Workflow übernommen. Er ruft Funktionen im Modul Files.pm auf, die einen String mit dem Inhalt der jeweiligen Datei zurück geben. Teilweise unterstützen die Funktionen Anpassungen, die vom Administrator vorgegeben werden können.

Im Verzeichnis $VAMOS/templates liegen eine Reihe von Vorlagen, die normalerweise in den Kopf der zu erstellenen Datei eingetragen werden. In $VAMOS/conf/files.conf existieren mehrere Sektionen, die mit dem Namen jeweils zur Generierungsfunktion passen.

Welchen Zweck die Parameter erfüllen ist hoffentlich selbsterklärend - sonst muß in der passenden Funktion im Code nachgesehen werden.

7 Bekannte Probleme

7.1 Der VAMOS Server

Wenn viele Anfragen gleichzeitig auf den Server einprasseln, bleiben manchmal Zombies zurück. Warum, weiß ich leider auch nicht. Vielleicht existiert das Problem aber im Net::Daemon Modul, welches nicht gut genug skaliert. Weil das aber nur relativ selten passiert und die Zombies eigentlich niemanden stören, habe ich mich darum noch nicht weiter gekümmert. In sehr seltenen Fällen raucht der Server teilweise vollständig ab (keine Verbindung mehr möglich, ...). In diesen Fällen müßte der Server mit Hilfe des Init-Skriptes neu gestartet werden.

7.2 Die MySQL Datenbank

Die zur Zeit verwendete Datenbank besteht ausschließlich aus InnoDB-Tabellen. Diese haben eine Eigenschaft, die sich konsistentes Lesen nennt. Das bedeutet, eine Session muß selbst eine Transaktion abschließen, um die Änderungen der anderen Sessions zu sehen. Dies kann zu einer Race Condition führen, da Änderungen einer Serverinstanz auf der Datenbank für die anderen Instanzen erst verspätet zu erkennen sind.

8 Abschließende Bemerkungen

Hier nun noch ein paar abschließende Bemerkungen. Ich habe mich bemüht ein CVS-Repository so gut es geht zu pflegen. Dieses enthält die wichtigsten Dateien und unter anderem auch diese Dokumentation. Um Änderungen nachvollziehbar zu halten, sollten diese ausschließlich im CVS unternommen werden. Das CVS-Root liegt unter /project/VAMOS/cvs-repo/. Ein „**cvs checkout vamos.rpc**“ müßte den Baum von vamos.rpc ins aktuelle Verzeichnis befördern. Sollten einige Dinge wider Erwarten trotzdem nicht dort enthalten sein, liegen sie bestimmt unterhalb meines Homedirectories unter ahaupt/vamos.rpc. Die ACLs sind für Mitglieder der Gruppe vamos für Lesezugriffe aufgedreht.

9 Anhang

9.1 Benötigte Perl-CPAN-Module

DBI	abstrakte Datenbankschnittstelle
DBD	spezifischer Datenbanktreiber – hier mysql
Config::IniFiles	einlesen der Konfiguration
PlRPC	Client / Server Implementierung
Net::Daemon	von PlRPC benötigt
IPC::Shareable	teilen von Daten unter geforkten Serverprozessen
Storable	umwandeln von Datenstrukturen in Datenströme
Compress::Zlib	Datentransfer komprimieren
String::Random	Generierung eines zufälligen Sitzungsschlüssels
Crypt::Blowfish	verschlüsselte Datenübertragung
Crypt::OpenSSL::RSA	Übertragung des Sitzungsschlüssels, RSA Authentisierung
Crypt::OpenSSL::Random	von Crypt::OpenSSL::RSA benötigt
Authen::Krb5	Kerberso5 Authentisierung
(Authen::)Krb4	Kerberos4 Authentisierung
Authen::PAM	PAM Authentisierung
Time::HiRes	Speichern des letzten Refreshs
Sys::Hostname	Ermittlung des Hostnamens
Mail::Send	Verschicken von E-Mails bei ernsthaften Problemen