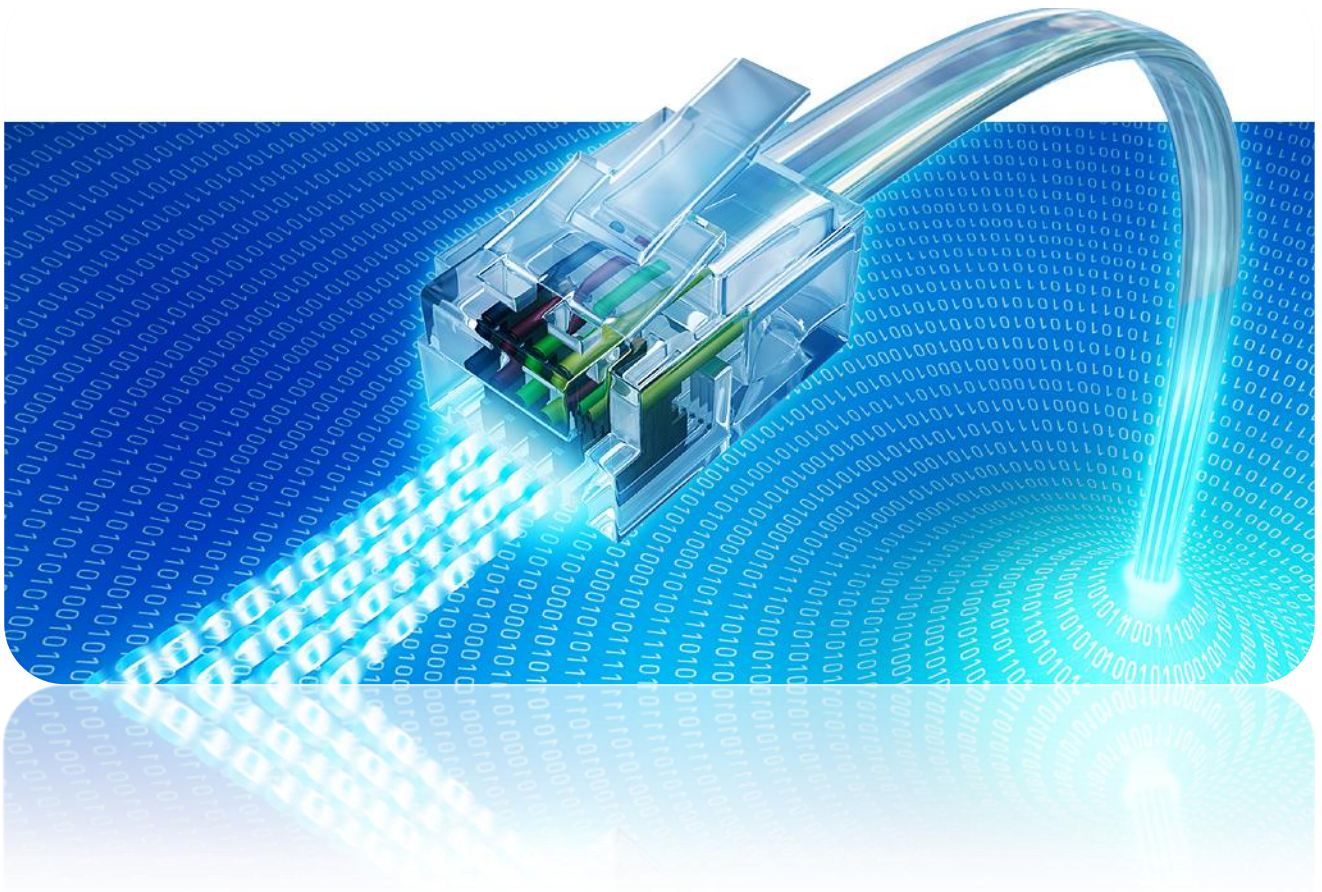


Semesterarbeit HS09

Captive Portal Load Generator



Autoren: Oliver Zürcher, Ymyr Osman
Betreuer: Michael Schneider
Projektpartner: -
Experte: Prof. Beat Stettler

Inhaltsverzeichnis

1. Abstract
2. Management Summary
3. Technischer Bericht
4. Persönliche Berichte
5. Glossar
6. Literaturverzeichnis
7. Projektplan
8. Anforderungsspezifikation
9. Analyse & Design
10. Installation
11. API
12. Testdokumentation
13. Anhang

Inhaltsverzeichnis

1.	Abstract	9
1	Abstract	10
2.	Management Summary	11
1	Ausgangslage	12
1.1	Vorgehen / Technologien	12
1.2	Ergebnisse	12
1.2.1	VMRemoteControl	12
1.2.2	LoadGenerator	12
1.2.3	Control-Center	12
1.3	Ausblick	13
3.	Technischer Bericht	14
1	Aufgabenstellung	15
2	Vorgehen	15
2.1	Analysen	15
2.2	Architektur	15
2.3	Ergebnisse	16
2.3.1	Übersicht	16
2.3.2	Virtualisierung	16
2.3.3	Charaktere	18
2.3.4	Control-Center	19
2.3.5	VMRemoteControl	19
2.3.6	LoadGenerator	20
3	Schlussfolgerung	20
4.	Persönliche Berichte	21
1	Oliver Zürcher	22
2	Ymyr Osman	23
5.	Glossar	24
1	A	25
2	B	25
3	C	25
4	D	25
5	E	26
6	F	26
7	G	26
8	H	26
9	I	26
10	J	27
11	K	27

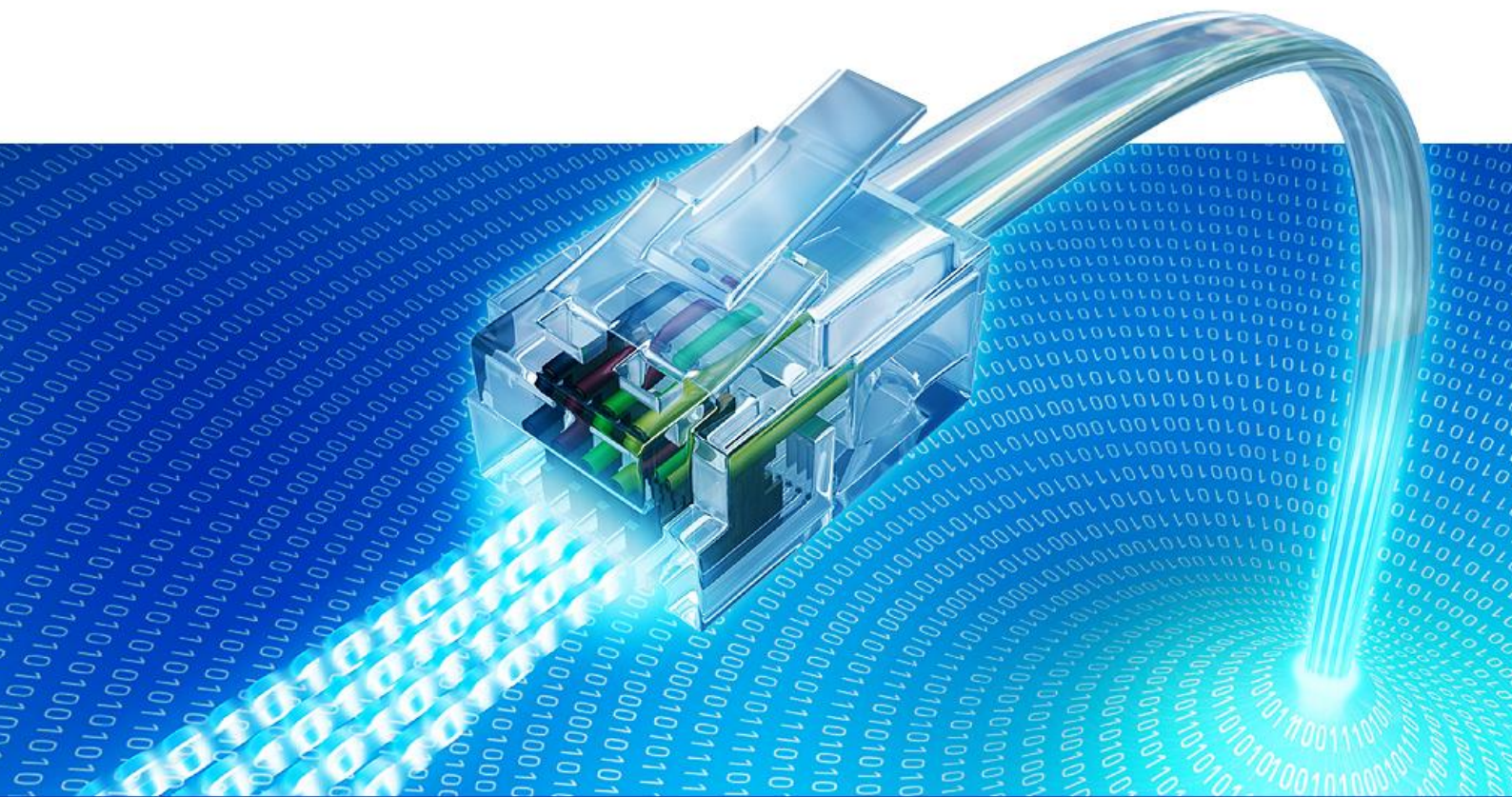
12	L	27
13	M	27
14	N	27
15	O	28
16	P	28
17	Q	28
18	R	28
19	S	28
20	T	29
21	U	29
22	V	29
6.	Literaturverzeichnis	30
1	Literaturverzeichnis	31
7.	Projektplan	32
1	Projektorganisation	33
1.1	Organisationsstruktur	33
1.2	Externe Schnittstellen	33
2	Management Abläufe	33
2.1	Projekt Kostenvoranschlag	33
2.2	Zeitplan	34
2.3	Meilensteine	35
2.4	Besprechungen	35
2.5	Releases	35
2.6	Artefakte	36
3	Risiko Management	37
4	Arbeitspakete	39
5	Infrastruktur	41
5.1	Räumlichkeiten	41
5.2	Hardware	41
5.3	Software	41
5.4	Backup	41
5.5	Kommunikation	41
6	Qualitätsmassnahmen	42
6.1	Dokumentation	42
6.2	Besprechungsprotokolle	42
6.3	Arbeitszeit	42
6.4	Versionskontrolle	42
7	Projektauswertung	43
7.1	Zeitauswertung	43
7.1.1	Zeitauswertung pro Iteration	43
7.1.2	Zeitauswertung pro Disziplin	44
7.1.3	Zeitauswertung pro Semesterwoche	44
7.2	Codeauswertung	45
7.2.1	Visual Studio Code-Auswertung	45
7.2.2	Code-Auswertung pro Projekt	46
7.2.3	Code-Auswertung gesamtes Projekt	46

8.	Anforderungsspezifikationen	47
1	Allgemein	48
1.1	Produktperspektive	48
1.2	Produkt Funktion	48
1.3	Benutzercharakteristik	48
1.4	Einschränkungen	48
1.5	Annahmen	48
1.6	Abhängigkeiten	48
2	Spezifische Anforderungen	49
2.1	Funktionalität	49
2.1.1	Erweiterbarkeit	49
2.2	Bedienbarkeit	49
2.2.1	Erlernbarkeit	49
2.2.2	Wiederherstellbarkeit	49
2.2.3	Fehlerbehandlung	49
2.3	Zuverlässigkeit	49
2.4	Leistung	49
3	Wartbarkeit	49
4	Schnittsteelllen	49
4.1	Benutzerschnittstelle	49
4.2	Softwareschnittstelle	49
4.3	Lizenzanforderungen	50
4.4	Verwendete Standards	50
4.5	Sicherheit	50
5	Use Cases	50
5.1	UC 1: Szenarios verwalten	50
5.2	UC 2: Szenarios Starten/Stoppen	51
5.3	UC 3: VM Image konfigurieren	51
9.	Analyse & Design	52
1	Analyse Virtualisierungssoftware	53
1.1	Anforderungen	53
1.2	Testumgebung / Testdefinition	53
1.3	VMWare	53
1.3.1	Grundlegendes	53
1.3.2	Test A: 1 VM	54
1.3.3	Test B: 50 VMs	54
1.4	VNUML	55
1.4.1	Grundlegendes	55
1.4.2	Test A: 1 VM	56
1.4.3	Test B: 50 VMs	56
1.5	QEMU	56
1.5.1	Grundlegendes	57
1.5.2	Test A: 1 VM	57
1.5.3	Test B: 50 VMs	57
1.6	Xen	57
1.6.1	Grundlegendes	57
1.6.2	Test A: 1 VM	57
1.6.3	Test B: 50 VMs	58

1.7	Kernel-based Virtual Machine	58
1.7.1	Grundlegendes	58
1.7.2	Test A: 1 VM	58
1.7.3	Test B: 50 VMs	59
1.8	Performance Test	59
1.9	Vergleich	61
2	Programmiersprache / Framework	63
3	Analyse Surfers	64
3.1	News-Leser	64
3.2	Googler	64
3.3	Downloader	64
3.4	MPPHammer	64
3.5	WebSiteTrasher	64
4	Prototyping Control Center	65
4.1	1 st Draft	65
4.2	2 nd Draft	66
4.3	3 rd Draft	67
4.4	Finaler Entwurf	68
5	MPP	69
5.1	Funktionsweise	69
6	Design & Architektur	70
6.1	Architektur Übersicht	70
6.2	Deployment	72
6.3	Services	72
6.3.1	IRegister - Registrationservice	72
6.3.2	IStatus - Statistik Reporting Service	73
6.3.3	IVMControl – VMs steuern und verwalten	73
6.4	Netzwerk / Kommunikation	74
6.4.1	Grundlegendes	74
6.4.2	Node Discovery	74
6.4.3	Beispielssequenzdiagramm	75
6.4.4	Startbefehl per Multicast	75
6.4.5	Kommunikation untereinander	75
6.5	Logische Architektur	76
7	KVM – Kernel based Virtual Machine	77
7.1	Images	77
7.2	Virtuelle Maschinen starten	77
8	Szenarien	78
8.1.1	Serialisierung	79
10.	Installation	80
1	Installation / Konfiguration	81
1.1	Control-Center	81
1.2	VMRemoteControl	81
1.2.1	Benötigte APT Packages	81
1.2.2	Netzwerkkonfiguration	82
1.2.3	Benötigte Assemblies	82
1.2.4	Installationsscript	82
1.3	Base-Image LoadGenerators	82

1.3.1	Benötigte APT Packages	82
1.3.2	Benötigte Assemblies	82
11.	API	84
1	Load Generator API	85
1.1	Schnittstelle	85
1.2	Contracts	86
2	Implementation	87
2.1	HTTP Request / Parsing von HTML Seiten	87
2.2	Umsetzung der Surfers	88
2.2.1	Googler	88
2.2.2	NewsReader	88
2.2.3	Downloader	89
2.2.4	MPPHammer	89
12.	Testdokumentation	90
1	Unit Tests	91
2	Systemtest	91
2.1	Durchlauf 1	91
2.2	Durchlauf 2	92
13.	Anhang	94
1	Installation VNUML unter Linux	95
1.1	Grundinstallation	95
1.2	Installation des Kernels	95
1.3	Installation des Dateisystems	95
1.4	Beispiel einer Ausführung	96
2	Linux Kernel kompilieren	96
2.1	Konfiguration	96
2.2	Kernel im Bootmanager (LILO) eintragen	97
3	Installation von Xen unter Linux	97
3.1	Basis Installation	97
3.2	Konfiguration einer Virtuellen Maschine mittels xen-tools	98
3.3	NAT-Konfiguration	98
3.4	GRUB-Bootloader Eintrag (Optional)	99
4	KVM-Installation	99
4.1	Basis Installation	99
4.2	Netzwerkeinrichtung	99
4.3	Klone erstellen und starten	100
5	Installationsscripts	101
5.1	Node	101
5.2	Base Image – Load Generator	104
6	get.sh	104
7	Quickstarterguide	106
7.1	Übersicht	106
7.1.1	Szenario Settings	106
7.1.2	Connected Nodes	106
7.1.3	Node Manager	107

7.1.4	Control	107
7.1.5	Statistics	107
7.1.6	Capacity Bar	107
7.2	Quickstart	107
7.3	Weitere Funktionen	109
7.3.1	Downloader	110
7.3.2	Googler	110
7.3.3	NewsReader	110
7.3.4	WebsiteTrasher	111
7.3.5	Control Center Settings	111
7.3.6	Capacity Bar	112
8	Sitzungsprotokolle	113
8.1	Woche 2	113
8.2	Woche 3	114
8.3	Woche 4	115
8.4	Woche 5	116
8.5	Woche 7 (MS Demo)	117
8.6	Woche 8	118
8.7	Woche 9	119
8.8	Woche 10	120
8.9	Woche 11	121
8.10	Woche 12	122
8.11	Woche 13	123
8.12	Woche 14	124
9	Abbildungsverzeichnis	125
10	Sourcecode Performance Messung	127



1. Abstract

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Abstract

Das INS hat 2003 für den Flughafen Zürich die Internet Access Plattform MPP entwickelt. Seither hat sich der Kundenstamm auf Hochschulen, Grossfirmen und Service-Provider vergrössert. Durch den grossen Zuwachs verkomplizieren sich die Testszenarios, da das MPP mit steigender Anzahl Clients zu testen ist. Die Anzahl Clients hat bereits seine Grenzen erreicht, so dass es nicht mehr möglich ist, „von Hand“ zu testen. Es soll nun eine Applikation zum effizienten Testen des MPPs entwickelt werden.

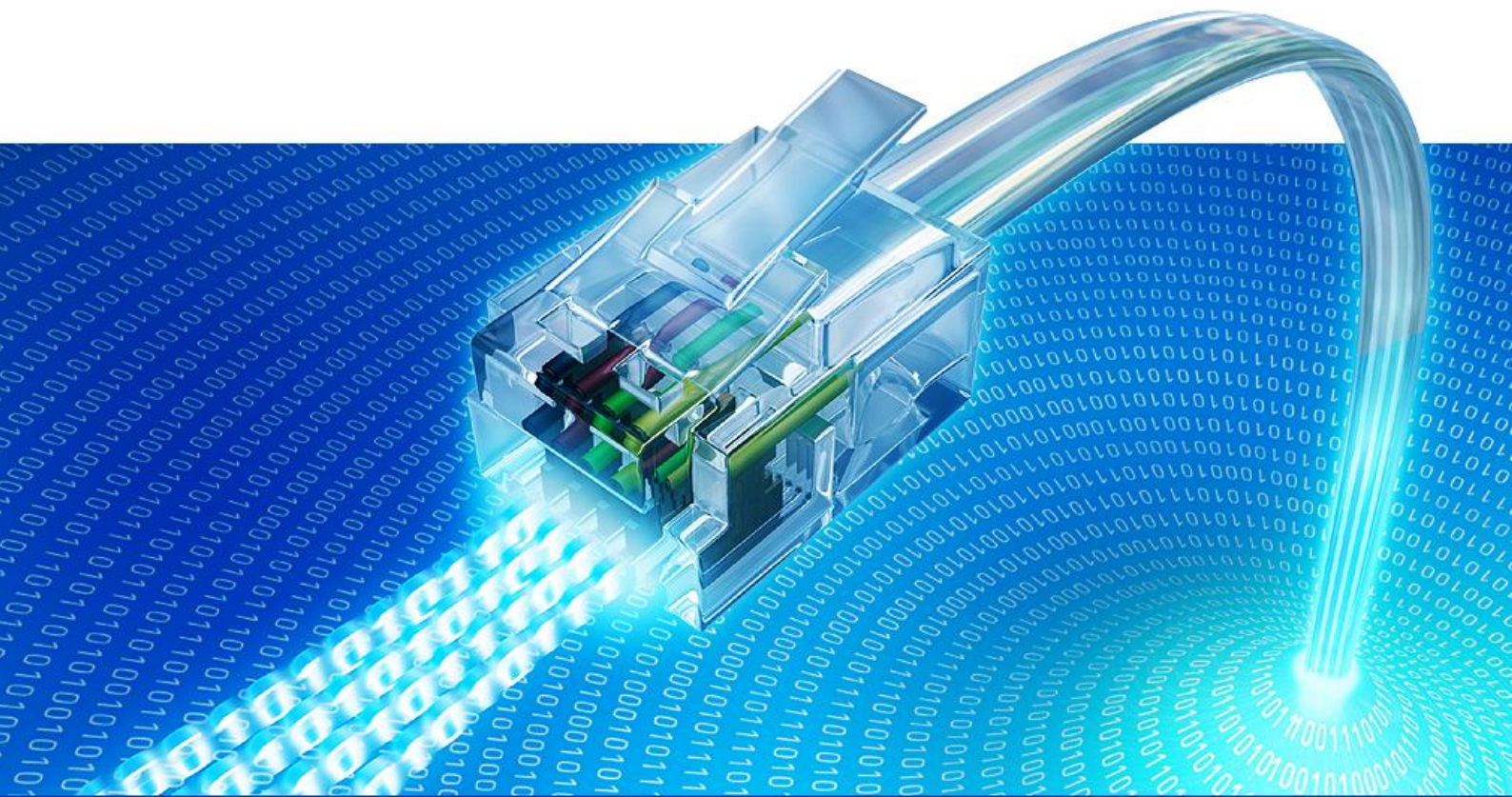
Als erstes wurden KVM, UML, XEN, VMWare und QEMU als Lösungen zur Virtualisierung der Clients getestet. Es galt dabei, möglichst wenige Ressourcen pro Client zu beanspruchen, und trotzdem muss die virtualisierte Umgebung genügend Performance aufweisen. Um die verschiedenen Lösungen zu vergleichen, wurde ein Performancetool entwickelt, welches die Leistung der virtuellen Umgebung misst.

Da ein Computer alleine nicht mehrere Tausend Clients simulieren kann, werden diese auf mehrere Computer verteilt (sogenannte Nodes). Um den administrativen Aufwand klein zu halten, wird das verteilte System von einem Punkt aus (Control-Center) gesteuert.

Pro virtuelle Maschine wird ein Client ausgeführt, welcher Last gegen das MPP generiert. In der Realität existieren verschiedene Charaktere, welche Last generieren. Eine Analyse definiert verschiedene mögliche Charaktere, auch Surfers genannt. Die Software ist so gebaut, dass später neue Charakter hinzugefügt werden können, ohne dass bestehende Teile der Software neu kompilieren werden müssen.

Für die Virtualisierung wird Kernel-based Virtual Machine (KVM) eingesetzt. Bei der Interprozesskommunikation (IPC) über die Computergrenze hinweg wird auf das ausgereifte .NET Remoting Framework zurückgegriffen.

Im Labor konnte der Prototyp mit 24 PCs à je 67 VMs (total 1600 VMs) gestartet werden. Per Multicast werden alle VMs angewiesen, mit dem generieren der Last zu beginnen. Im Control-Center wird der Fortschritt sowie auftretende Fehler angezeigt.



2. Management Summary

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Ausgangslage

Das INS hat 2003 für den Flughafen Zürich die Internet Access Plattform MPP entwickelt. Seither hat sich der Kundenstamm auf Hochschulen, Grossfirmen und Service-Provider vergrössert. Auch die Anzahl Clients ist an diesen Hotspots in den letzten Jahren regelrecht explodiert, was zu einem sehr starken Anstieg der Last auf den Portalen geführt hat.

Durch den grossen Zuwachs verkompliziert sich die Durchführung von Tests gegen das MPP. Die Anzahl Clients für ein in der Realität entsprechendes Szenario ist bereits zu gross, um solche Tests von Hand effizient durchführen zu können.

Die Aufgabe besteht nun darin, eine Software zu entwickeln, welche es ermöglicht, tausende Clients zu simulieren, um realitätsnahe Szenarien für Tests zu ermöglichen.

1.1 Vorgehen / Technologien

Es existieren viele verschiedene Produkte zur Virtualisierung auf dem Markt. Es wurden 5 bekannte Lösungen ausgewählt (VMWare, QEMU, Xen, KVM und UML), um sie in einer detaillierten Analyse genauer zu betrachten. Bei möglichst wenig Ressourcenbelastung möglichst viel Leistung aufweisen war eines der wichtigsten Auswahlkriterien.

In einer weiteren Analyse wurden die verschiedenen Charaktere, welche sich am MPP einloggen und Last generieren analysiert und definiert.

Nach diesen beiden Analysen wurde die Software Architektur festgelegt. Es handelt sich um ein verteiltes System, welches von einem Punkt aus gesteuert werden kann.

Als Programmiersprache wurde C# mit dem .NET Framework von Microsoft gewählt. In Kombination mit dem Mono-Projekt lassen sich .NET Assemblies unter Linux ausführen.

1.2 Ergebnisse

Das verteilte System besteht aus verschiedenen Komponenten, welche von einem Punkt aus gesteuert werden (Control-Center):

1.2.1 VMRemoteControl

Die Konsolenanwendung VMRemoteControl läuft auf jedem Computer (Node), welcher seine Hardware für die Testszenarios zur Verfügung stellt. Die Applikation bietet Remote-Services zur Verwaltung und Steuerung von virtuellen Maschinen auf dem System. Für jede VM wird das vorkonfigurierte Basisimage verwendet.

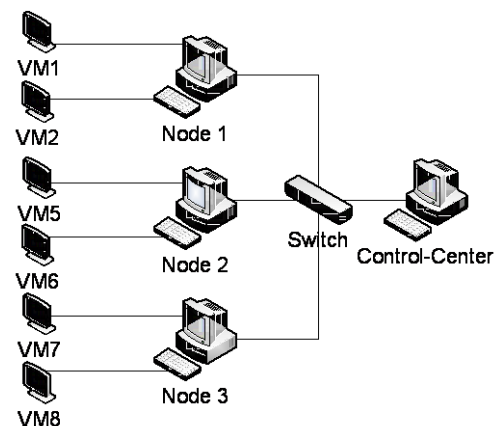
1.2.2 LoadGenerator

Die LoadGenerator Anwendung ist ebenfalls eine Konsolenanwendung. Sie ist im Basisimage für die virtuellen Maschinen vorinstalliert und wird beim Systemstart automatisch gestartet. Das Control-Center weist dem LoadGenerator einen Surfertypen zu, welcher dynamisch zur Laufzeit nachgeladen wird. Dieser generiert nach dem Startsignal vom Control-Center die Last gegen das MPP.

1.2.3 Control-Center

Das Control-Center ist eine GUI Applikation, welche dem Benutzer eine Eingabemaske zur Verwaltung der verschiedenen Szenarien zur Verfügung stellt.

Zur Findung aller Nodes im Netzwerk, welche für die Virtualisierung der Clients zuständig sind,



werden per Multicast alle Nodes getriggert, sich beim Control-Center anzumelden. Diese teilen bei der Registrierung ihre verfügbaren Ressourcen dem Control-Center mit. Wenn genügend Ressourcen für das konfigurierte Szenario zur Verfügung stehen, kann das Szenario im Control-Center gestartet werden. Es kann wahlweise eine Anzahl an Durchläufen konfiguriert werden, oder fortlaufend, bis der Benutzer das Szenario stoppt.

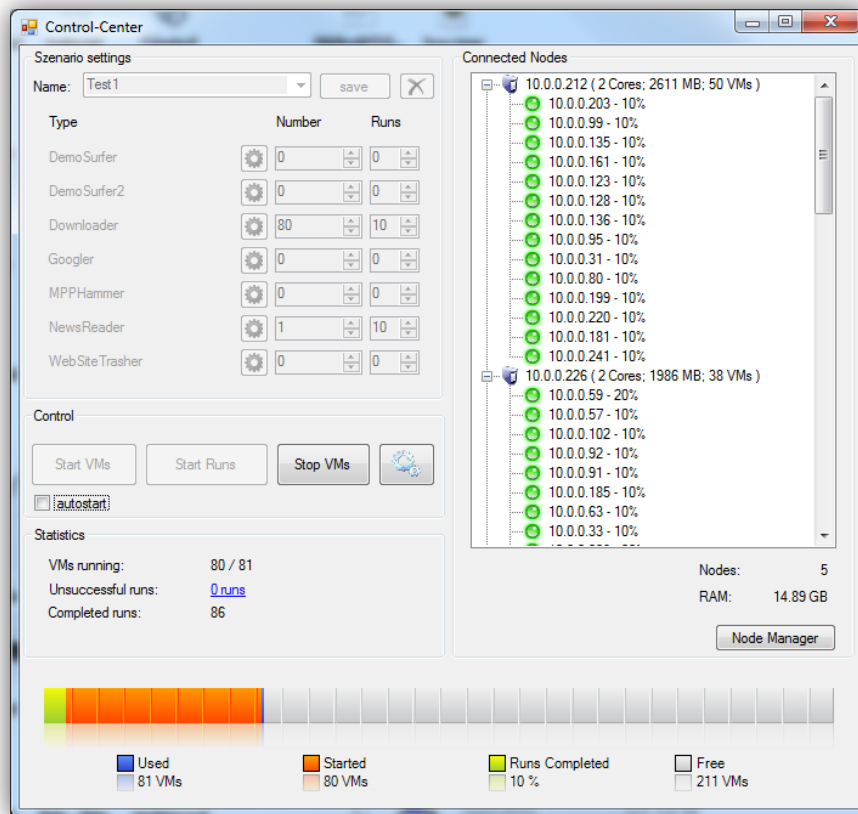
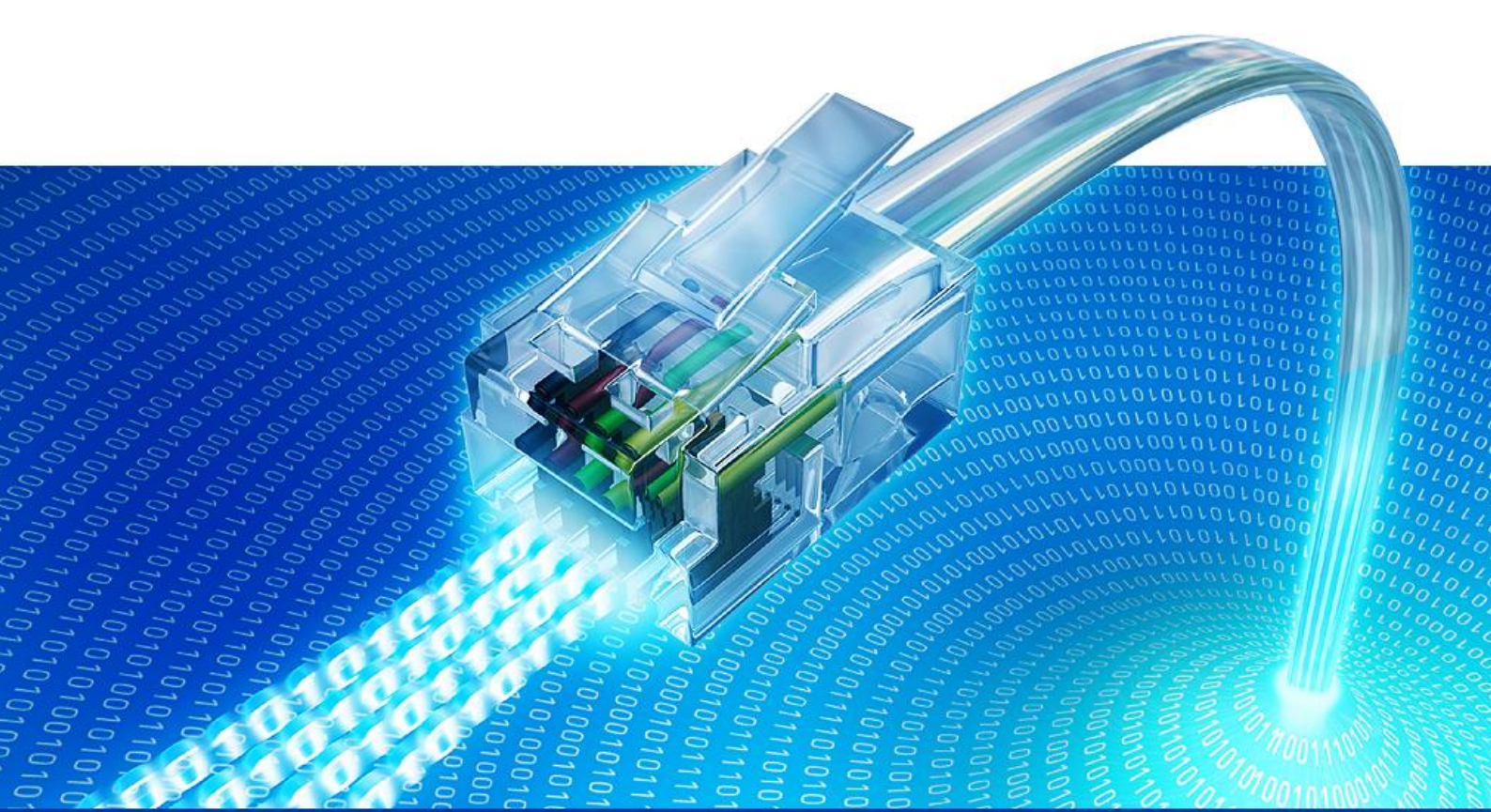


Abbildung 1: Control-Center GUI

1.3 Ausblick

Die entwickelte Software ist ein lauffähiger Prototyp. Durch die Load Generator API können weitere Surfertypen entwickelt werden. Die Architektur ist so aufgebaut, dass diese dynamisch geladen werden. Bestehender Code muss beim Hinzufügen weder geändert noch neukompiliert werden. Das Captive Portal Load Generator Projekt bietet eine hervorragend skalierende Lösung zum effizienten Testen des MPPs.



3. Technischer Bericht

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Aufgabenstellung

Das INS hat 2003 für den Flughafen Zürich die Internet Access Plattform Multi Provider Portal (MPP) entwickelt. Seither hat sich der Kundenstamm auf Hochschulen, Grossfirmen und Service-Provider vergrössert. Auch die Anzahl Clients ist an diesen Hotspots in den letzten Jahren regelrecht explodiert, was zu einem sehr starken Anstieg der Last auf den Portalen geführt hat.

Durch den grossen Zuwachs verkompliziert sich die Durchführung von Tests gegen das MPP. Die Anzahl Clients für ein in der Realität entsprechendes Szenario ist bereits zu gross, um solche Tests von Hand effizient durchführen zu können.

Die Aufgabe besteht nun darin, eine Software zu entwickeln, welche es ermöglicht, tausende Clients zu simulieren, um realitätsnahe Szenarien für Tests zu ermöglichen.

2 Vorgehen

2.1 Analysen

Um mehrere Tausend Clients zu simulieren, reicht ein Computer alleine nicht aus. Damit bereits bestehende Computer in den Schulzimmern verwendet werden können, wurde als Architektur ein verteiltes System geplant, welches durch das hinzufügen weiterer Computer einfach erweitert werden kann. Diese sollen von einem Punkt aus (Control-Center) über das Netzwerk gesteuert werden. Auf den Computer, sogenannte Nodes, sollen für jeden simulierten Client eine VM gestartet werden. Dazu mussten in einer ersten Analyse verschiedene Virtualisierungslösungen analysiert werden. Das Ziel dieser Analyse war, die beste Lösung für maximale Performance bei möglichst geringer Ressourcenbelastung zu finden, damit möglichst viele VMs pro Node gestartet werden können. Dabei spielten aber Faktoren wie der administrative Aufwand für die Erstellung der VM ebenfalls eine wichtige Rolle, da pro Computer mit sicher mehr als 50 VMs gerechnet wurde.

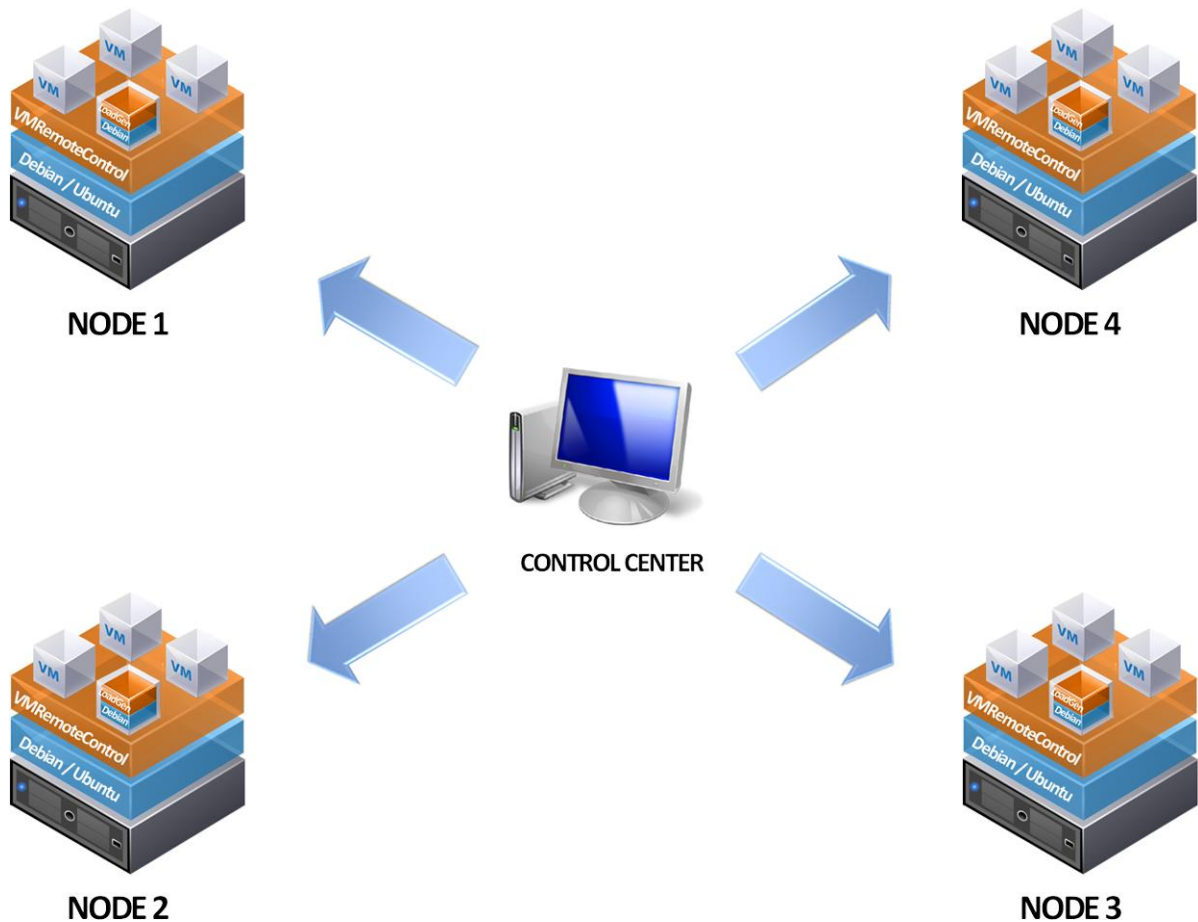
In der Realität existieren verschiedene Charaktere, welche sich am MPP anmelden und Last generieren. Damit möglichst der Realität entsprechende Szenarien getestet werden können, wurden in einer zweiten Analyse die verschiedenen Charaktere mit ihren Eigenschaften definiert.

2.2 Architektur

Wie bereits erwähnt wurde ein verteiltes System als Architektur gewählt. Auf einem Computer ist das Control-Center installiert, über welches alle anderen Computer (Nodes) gesteuert werden können. Das Control-Center gibt Anweisungen zum Erstellen, Starten und Stoppen von virtuellen Maschinen, welche die virtuellen Clients simulieren.

2.3 Ergebnisse

2.3.1 Übersicht



2.3.2 Virtualisierung

Bei der Analyse wurden die 5 Produkte XEN, QEMU, VMWare, UML und KVM getestet. Nicht alle Produkte sind dafür ausgelegt, tausende VMs zu verwalten. Bei VMWare zum Beispiel müssen alle VMs einzeln mit mehreren Mausklicks einer Storage hinzugefügt werden – ein relativ grosser Aufwand für tausend VMs. Die CLI unterstützt einem erst nach dem Hinzufügen zur Storage. Damit fiel VMWare als Option weg. Die anderen Produkte hingegen konnten die Anforderung an die Verwaltbarkeit alle erfüllen.

Neben den benötigten Ressourcen und der Verwaltbarkeit zählt die Ausführungs geschwindigkeit ebenfalls zu den wichtigen Punkten der Anforderung: Je schneller sie ist, desto besser skaliert die Lösung.

Die Ausführungs geschwindigkeit ist grundlegend von zwei Faktoren abhängig: Die Geschwindigkeit der CPU und dem Datendurchsatz im RAM. Um diese beiden Faktoren vergleichen zu können, wurde ein Stressprogramm entwickelt, welches im Anhang zu finden ist. Es berechnet parallel mit 2 Threads die Wurzeln von 1 bis 100'000'000, anschliessend werden 10 mal hintereinander 5/20 MB im RAM linear kopiert. Die benötigte Zeit für die Operationen ist in folgenden 2 Diagrammen ersichtlich:

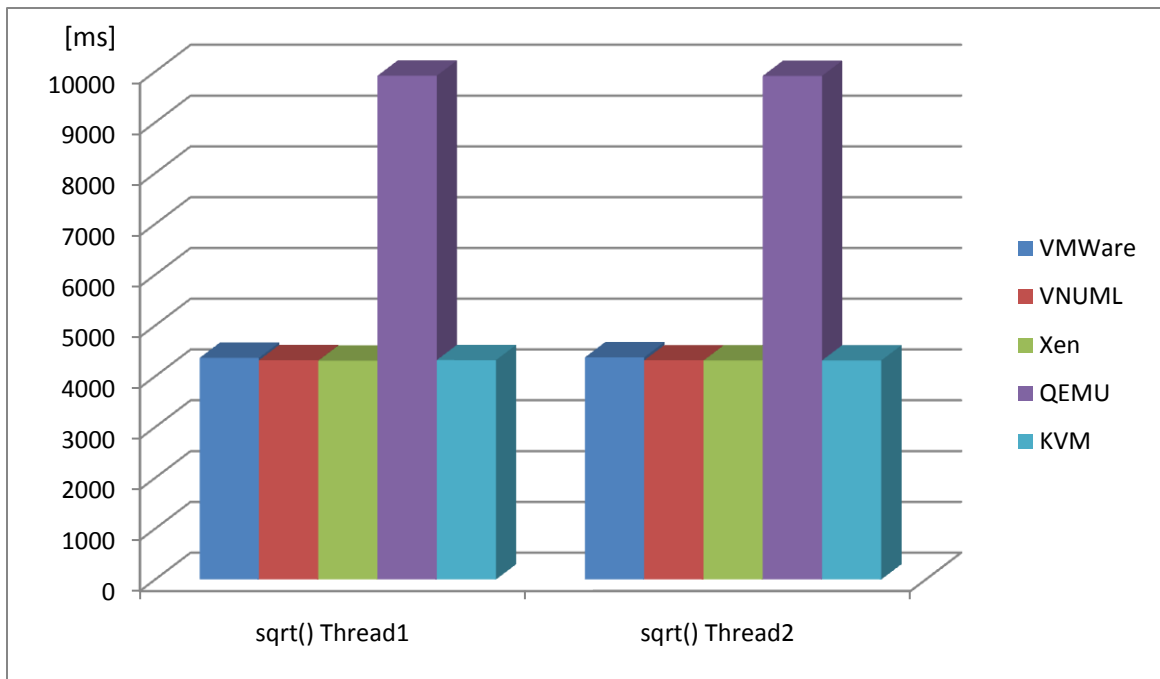


Abbildung 2: Zeitmessung sqrt()

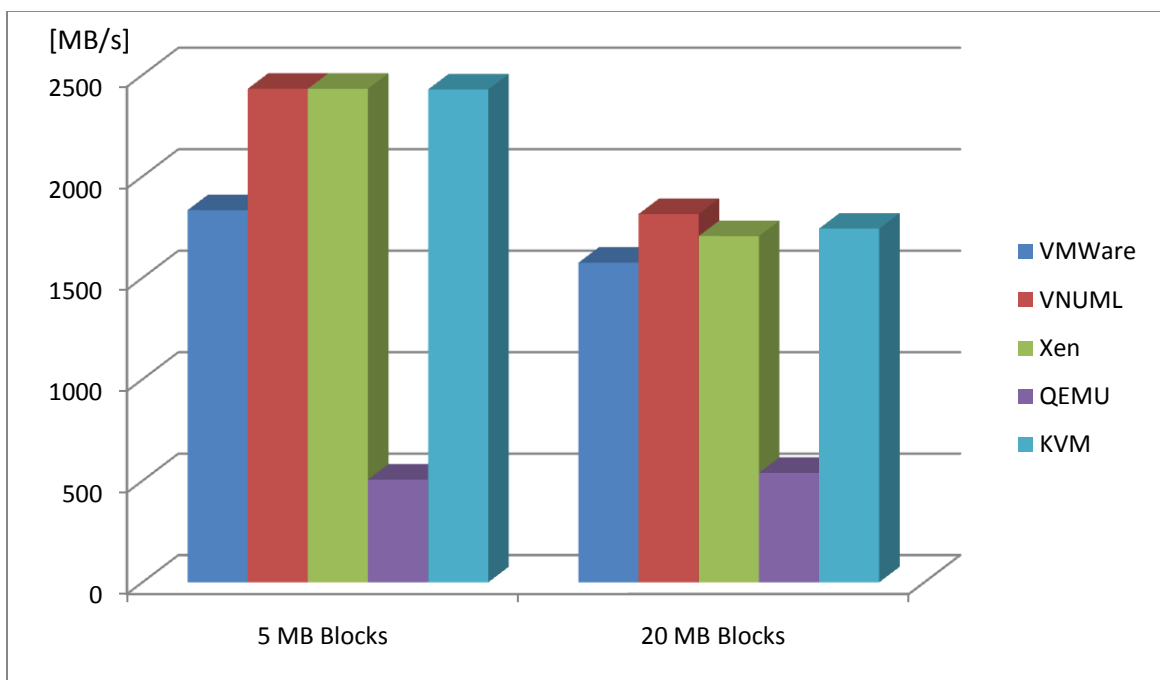


Abbildung 3: Memory Test

Der Performancetest fiel sehr gleichmässig aus, lediglich QEMU zeigte deutlich seine Schwäche.

VMWare ist durch den hohen Verwaltungsaufwand keine gute Wahl. Des Weiteren hat VMWare Mühe, wenn ein Image mehr als 100 Mal geklont wird.

VNUML hat durch die Performance und vor allem der sehr gute Verwaltbarkeit überzogen.

Leider können VMs nur sequentiell vom Parser gestartet werden, was bei mehreren VMs doch einige Zeit in Anspruch nehmen kann. Übrig bleiben Xen und KVM. Xen benötigt für das Basissystem 7 MB weniger RAM als KVM. Allerdings fehlt die Klonfunktion ganz. 7 MB RAM pro

VM mehr sind ein guter Kompromiss, welcher für die zusätzliche Klonfunktion von KVM eingegangen werden kann.

KVM schloss sehr gut im Performancetest ab. Sämtliche Einstellungen lassen sich über die CLI einstellen und sind gut dokumentiert. Des Weiteren überzeugen die Overlay-Images sehr. Sie sind einfach verwaltbar, sparen viel Speicherplatz ein und sind sehr schnell erstellt. Aus diesen Gründen wird KVM für die Virtualisation eingesetzt.

2.3.3 Charaktere

In der Analyse wurden 5 verschiedene Charaktere definiert, welche sich am MPP einloggen:

- **News-Leser**
Der News-Leser loggt sich beim MPP ein. Danach liest er einen Newsartikel von einer Newsseite und meldet sich gleich wieder vom MPP ab. Dadurch wird durch die vielen Sessions, welche auf- und abgebaut werden, das MPP belastet.
- **Googler**
Ein Googler ist jemand, der nach Begriffen sucht und relativ schnell von Seite zu Seite wechselt, bis er seine gesuchte Information gefunden hat. Dabei kann er mehrere Google Resultate schnell absuchen (und bei denen ein wenig in die Tiefe hineingehen – wenn nicht fündig nächstes Google Resultat absuchen). Der Aufenthalt auf einer Seite ist wesentlich kleiner als beim News-Leser (ca. 10 Sekunden pro Seite bis zum Ziel). Der Unterschied zum News-Leser besteht darin, dass er sich nicht so oft beim MPP an- und abmeldet.
- **Downloader**
Der Downloader lädt eine Datei von einer URL herunter. Danach meldet er sich wieder ab.
- **MPPHammer**
Der MPPHammer meldet sich beim MPP nicht an. Er versucht trotzdem verschiedene Seiten aufzurufen. Das MPP leitet ihn jedesmal auf die Landing Page um.
- **WebSiteTrasher**
Der WebSiteTrasher macht mehrere Zugriffe asynchron auf eine Webseite. Den WebSiteTrasher gibt es nicht in der Realität, er ist zum Testen von Webserver gedacht, da er pro Sekunde extrem viele Requests abschicken kann.

2.3.4 Control-Center

Das Control-Center ist die Schnittstelle vom Anwender und dem ganzen System. Es ermöglicht dem Benutzer, verschiedene Szenarien zu definieren und zu verwalten. Die Nodes melden sich beim Control-Center an und stellen damit ihre Hardware dem Control-Center zur Verfügung. Wenn ein Szenario gestartet wird, weist das Control-Center den Nodes VMs zu und startet diese Remote. Dadurch bleibt das System sehr flexibel und kann nach Belieben durch Hinzufügen weiterer Nodes vergrößert werden.

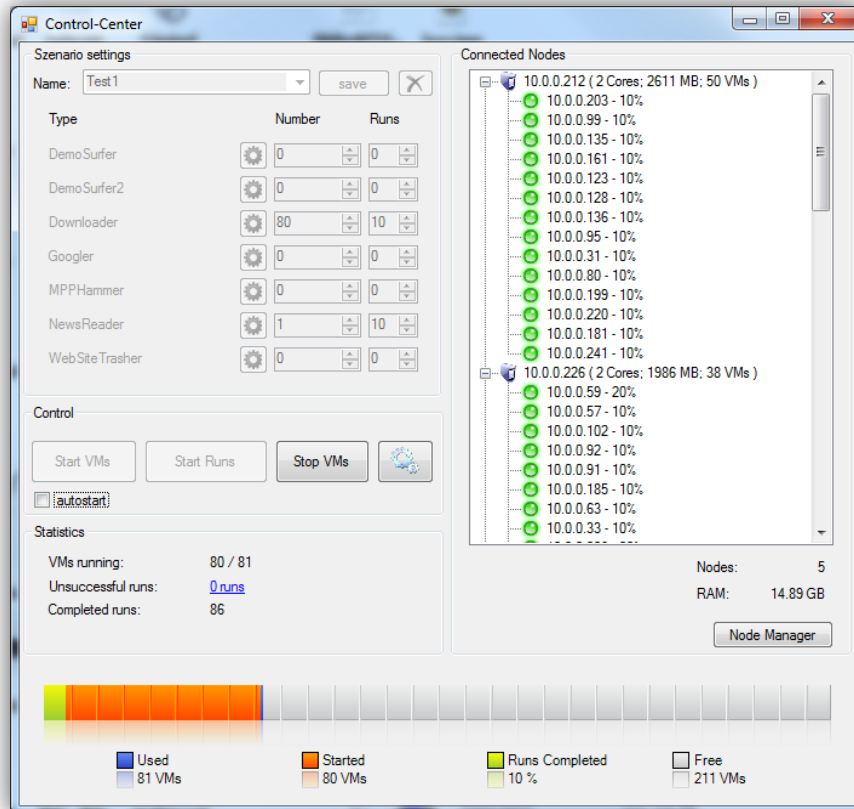


Abbildung 4: Control-Center GUI

2.3.5 VMRemoteControl

Die Konsolenapplikation VMRemoteControl wird auf jeder Node ausgeführt. Sie kann auf einer Multicast Adresse getriggert werden, sich beim Control-Center (Absender der Multicast-Nachricht) zu registrieren. Sie teilt bei der Registration dem Control-Center die Anzahl Cores und der verfügbare Arbeitsspeicher mit. Nach der Registration stellt die Anwendung dem Control-Center einen Service zur Steuerung und Verwaltung der virtuellen Maschinen auf der Node zur Verfügung:

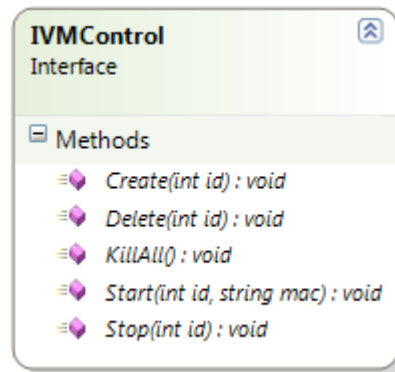


Abbildung 5: Remote Service Interface

Für die VMs wird ein Base-Image verwendet, welches auf alle Nodes verteilt werden muss. Es ist eine Debian Lenny Installation, bei der alle nötigen Konfigurationen bereits vorgenommen wurden. Für jede virtuelle Maschine wird ein Overlay-Image zu diesem Image gemacht. Das heisst, es wird das Base-Image verwendet, wenn aber Daten geändert oder neu hinzukommen, werden diese in das Overlay-Image gespeichert. Dies hat den Vorteil, dass eine neue VM in weniger als einer Sekunde erstellt werden kann (Overlay-Image ist initial 4 KB gross) und gleichzeitig viel Speicherplatz gespart werden kann. Das Klonen einer Linux Installation stellt überhaupt keine Probleme dar, lediglich die MAC-Adresse muss „unique“ sein. Diese kann beim Starten einer VM als Parameter angegeben werden.

2.3.6 LoadGenerator

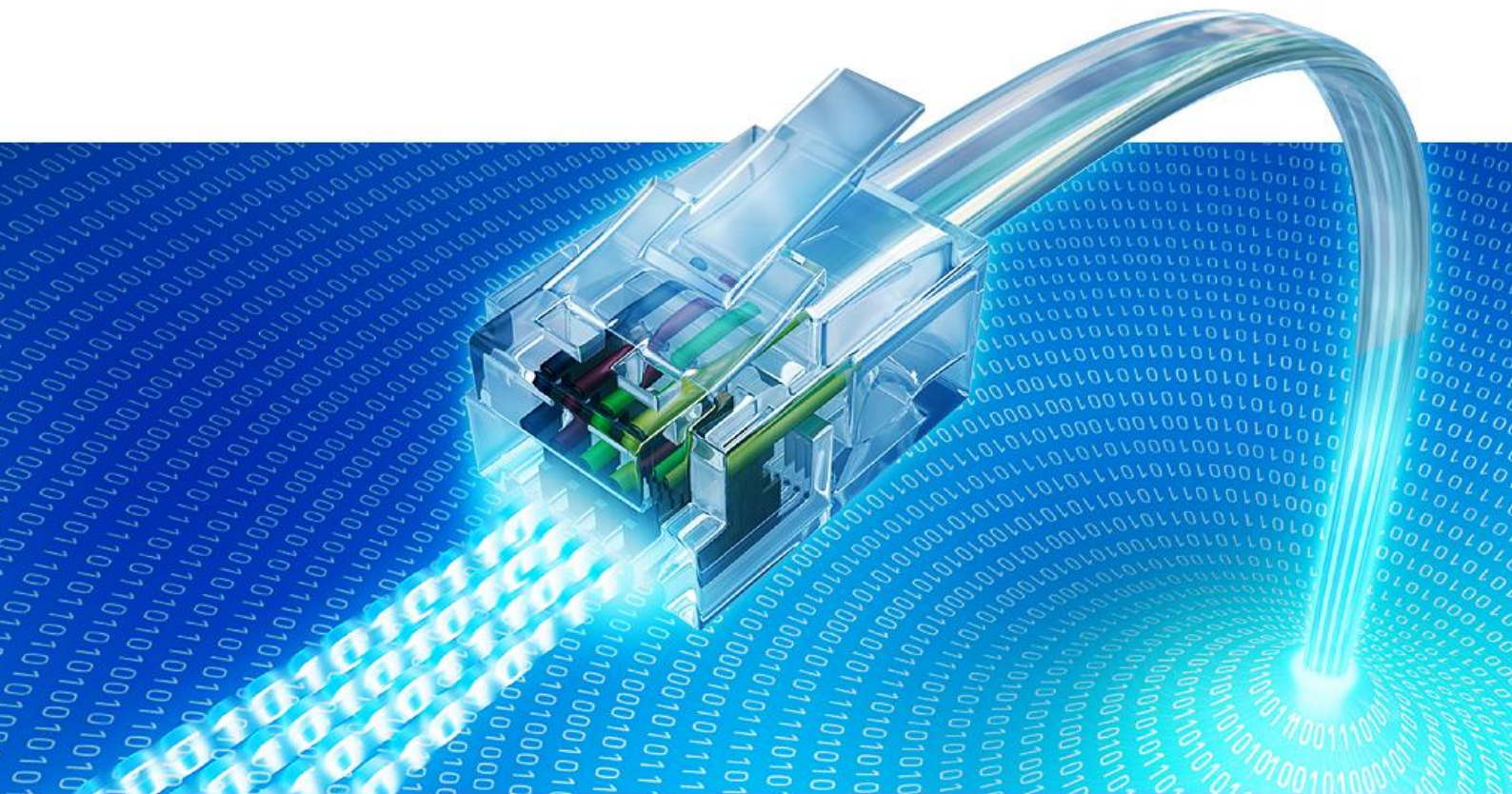
Die LoadGenerator Anwendung ist im VM-Image, welches sich auf den Nodes befindet bereits installiert. Nachdem das Netzwerkinterface „up“ ist, wird es automatisch gestartet (via ifup-script). Als Parameter nimmt die Anwendung eine IP oder DNS entgegen, bei welcher sie sich meldet, wenn sie startet. Die IP ist die vom Control-Center, welches der LoadGenerator Applikation daraufhin einen Surfertypen und dessen Einstellungen mitteilt. Dieser Typ wird per Reflection im aktuellen Verzeichnis in den DLL-Dateien gesucht und nachgeladen. Danach wartet die Anwendung auf das „go“ (Multicast-Nachricht) vom Control-Center, bevor es mit dem Generieren der Last beginnt. Während dem Lastgenerieren, sendet die Applikation dem Control-Center laufend Nachrichten über den Fortschritt.

3 Schlussfolgerung

Ziel der Arbeit war, ein lauffähiger Prototypen vorzuweisen, welcher gleichzeitig ca. 2000 Clients simulieren kann. Obwohl maximal 1600 Clients simuliert wurden im Praktikumsraum, wurde das Ziel erreicht. Unsere Analysen zeigen auf, dass die 24 Computer im Praktikumsraum nicht reichen, um 2000 Clients zu simulieren. Durch das Hinzufügen weiterer Computer kann diese Grenze aber ohne Probleme überschritten werden, denn der Prototyp skaliert problemlos.

Der Einsatz eines ausgereiften Frameworks für die „remote procedure calls“ hat sich sehr gut ausbezahlt. Eine Eigenimplementation hätte wesentlich mehr Zeit in Anspruch genommen. Die Monoimplementation des .NET Frameworks für Linux funktioniert auch mit dem .NET Remoting Framework hervorragend.

Der Prototyp zeigt, dass die verwendete Architektur funktioniert und für Tests gegen das MPP verwendet werden kann. Durch die API können Erweiterungen problemlos in die Applikation integriert werden.



4. Persönliche Berichte

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Oliver Zürcher

Bei der Auswahl der Arbeit für die Semesterarbeit war mein primäres Ziel, eine interessante Arbeit mit dem .NET Framework zu finden, um weitere Erfahrungen für das spätere Berufsleben zu sammeln. Leider gab es auf die .NET Arbeiten einen relativ grossen Ansturm, weshalb ich meine Wünsche etwas anpassen musste. Da ich selbst Interesse an Linux habe, erweiterte ich meine Wunschliste mit Arbeiten im Linux Gebiet. Dies allerdings mit etwas Vorsicht, da wir beide im Team keine Linux Cracks sind, aber doch ein paar Erfahrungen damit sammeln konnten und unseren Horizont gerne erweitern.

Mit der Zuweisung der Arbeit „Captive Portal Load Generator“ war ich sehr zufrieden und freute mich auf die Arbeit. Virtualisation wird ein immer wichtigeres Thema in der heutigen Zeit, um bestehende Ressourcen möglichst optimal auszulasten.

Im Nachhinein würde ich die Analyse der Virtualisation kürzer gestalten, da ziemlich wenig Zeit danach für die Implementation zur Verfügung stand und wir eigentlich beide gerne noch mehr entwickelt hätten. Die durchdachte Architektur macht es möglich, die Implementation der Virtualisation in relativer kurzer Zeit auf ein anderes Produkt zu wechseln. Unsere Erkenntnisse nach 3 Wochen (von total 6) hätten gereicht, um eine gute Wahl zu treffen. Diese Wahl wäre wahrscheinlich nicht auf KVM sondern auf XEN oder UML gefallen. Die Möglichkeit, die Implementation der Virtualisierung in weniger als einer Stunde zum Beispiel von KVM auf XEN zu wechseln, rechtfertigt in meinen Augen die 3 extra Wochen für die Analyse nicht. Trotzdem hat die Analyse wertvolle und interessante Erkenntnisse hervorgebracht.

Die Entscheidung .NET zu verwenden, hat sich in meinen Augen sehr gut auszubezahlt. Durch unser bestehendes Know-How war nur wenig Einarbeitungszeit in neue Technologien (.NET Remoting) nötig. Zu Beginn waren wir uns nicht ganz so sicher, wie gut Mono unter Linux funktionieren wird. Zu unserer Überraschung, funktionierten die Anwendungen auf Anhieb unter Linux. Performancetechnisch hinkt die Monoimplementation der Implementation von Microsoft zwar noch hinterher, jedoch kam dies in diesem Projekt nicht zum tragen.

Am Ende des Projektes schaue ich auf einen überaus positiven Projektverlauf zurück. Die Zusammenarbeit mit meinem Partner war wie in bisherigen Projekten sehr gut verlaufen. Auch die Arbeit mit den Betreuern, welche stets mit einer offenen und ehrlichen Art Feedback gaben, hat zu einem sehr guten Ergebnis aus meiner Sicht beigetragen.

2 Ymyr Osman

Im Rahmen der Studienarbeit hatte ich die Möglichkeit mich über eine längere Zeitspanne mit einer Arbeit intensiv zu beschäftigen. Diese Erfahrung ist sehr positiv für mich und motiviert auf die kommende Bachelor Arbeit.

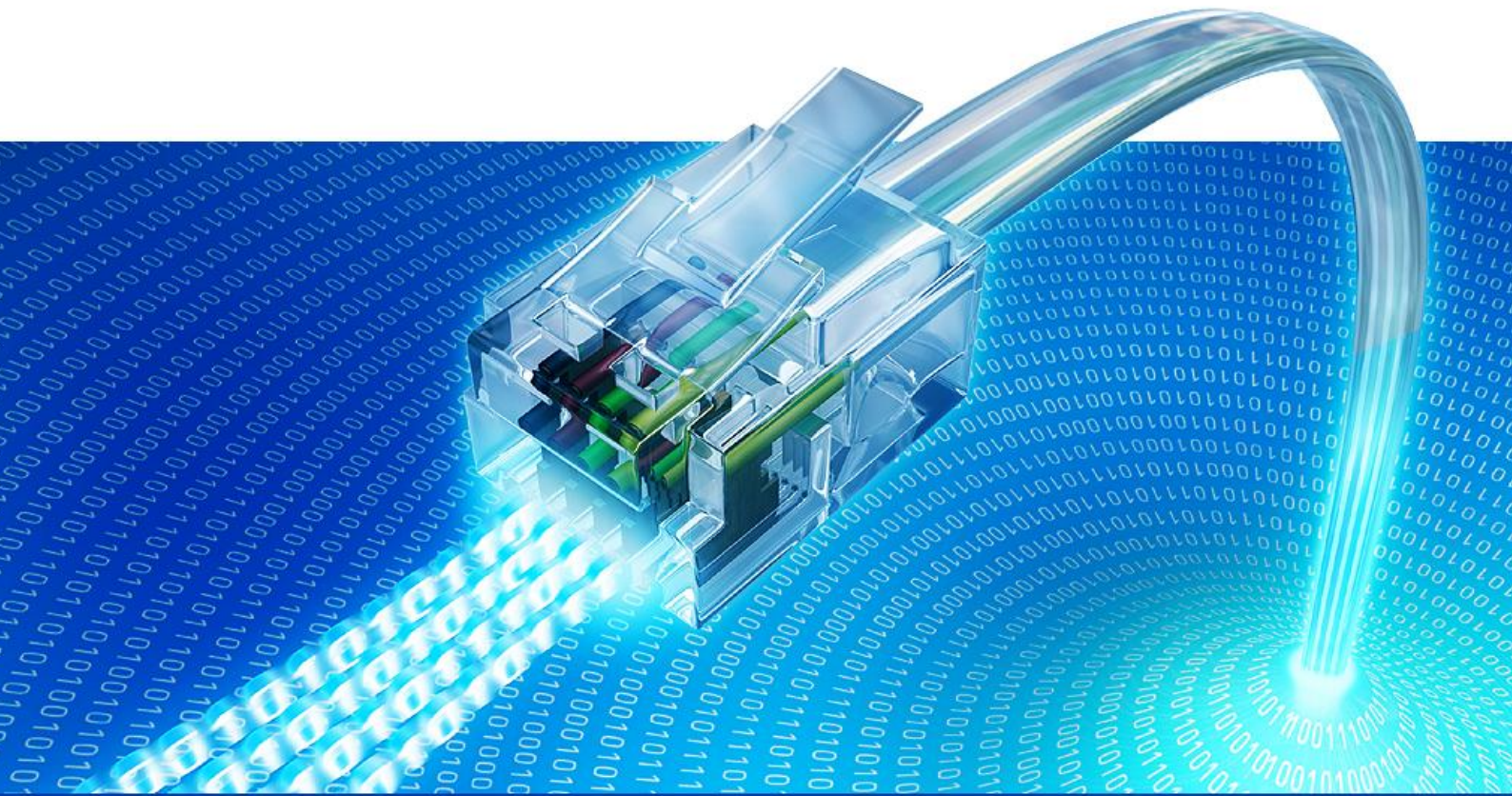
Besonders angenehm war die Arbeit mit meinem Partner. Zusammengearbeitet wurde schon im Software Engineering Projekt. Daher kannten wir uns bereits von der Arbeitsweise her, und es gab keine Probleme bezüglich Kommunikation oder Arbeitsteilung. Nicht nur die Teamarbeit war angenehm sondern auch die mit den Betreuern. Diese standen für uns immer zur Verfügung und haben uns in wichtigen Punkten geholfen und geführt.

Erfahrungen konnte ich im Bereich Linux sammeln. Durch das Projekt lernte ich die ganze Umgebung kennen. Interessant war auch zu sehen, wie weit das Mono-Projekt fortgeschritten ist. Bis auf wenige Probleme brauchte es kaum Umstellungen zur üblichen Entwicklung von .NET Applikationen.

Interessant war auch, das MPP kennen zu lernen. Bisher kannte ich das nur von den Orten, wo es eingesetzt wird. Wie es funktioniert und was es bietet war mir unbekannt.

Ein Punkt der besser gelöst werden soll für die Zukunft ist das Deployment der Applikation. Leider hat die Zeit nicht mehr gereicht, um sich eine bessere Lösung für die Installation und das Deployment zu erarbeiten.

Gesamthaf war das Projekt eine gute Erfahrung. Das bisher gelernte konnte wieder aufgefrischt und angewendet werden. Das neu gelernte ist für mich wertvoll, denn ich kann es für die Zukunft mitnehmen. Speziell wenn es heisst eine Applikation soll für Linux entwickelt werden, weiss ich, dass diese mittels dem .NET Framework umgesetzt werden kann.



5. Glossar

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 A

Begriff	Erklärung
API	Application Programming Interface => Programmierschnittstelle
APT	Advanced Packaging Tool ist ein Paketverwaltungssystem im Debian Betriebssystem
Assembly	Übersetzt Programmklassen als ausführbares Programm

2 B

Begriff	Erklärung
BSD	Berkley Software Distribution

3 C

Begriff	Erklärung
Client	Programm welches mit einem Server kommuniziert um Dienstleistungen zu nutzen
CLG	Captive Load Generator
Concurrency	Nebenläufigkeit, d.h. Zwei Threads erledigen Arbeiten ohne Abhängigkeiten
Control Center	CLG Hauptapplikation
COW	Copy On Write
cron	Jobsteuerung in Unix Betriebssystemen
CSS	Cascading Style Sheet ist eine deklarative Gestaltungssprache für strukturierte Dokumente.

4 D

Begriff	Erklärung
debootstrap	Ermöglicht eine völlige Neuinstallation eines Debian-Basisystems über das Netzwerk
DHCP	Dynamic Host Configuration Protocol ist ein Protokoll welches die Zuweisung der Netzwerkkonfiguration durch einen Server ermöglicht
DLL	Dynamic Link Library, bezeichnet allgemein

	eine Dynamische Bibliothek
--	----------------------------

5 E

Begriff	Erklärung
exim4	Mailserver

6 F

Begriff	Erklärung
Framework	Programmiergeüst

7 G

Begriff	Erklärung
Google	Bekannte Suchmaschine, www.google.com

8 H

Begriff	Erklärung
Host	Beherbergt Clients oder Server
HSR	Hochschule Rapperswil
HTML	Hypertext Markup Language ist eine Sprache, welche zur Strukturierung von Inhalten wie, Bildern, Texten oder Verweisen in Dokumenten dient.
HTTP Request	Anforderung an eine Web Ressource (HTTP)
HTTP Response	Gibt eine Antwort von einer Internetressource zurück
HW	Hardware

9 I

Begriff	Erklärung
Image	Speicherabbild, z.B. einer kompletten Festplatte
INS	Institute for networked solutions
Interface	Schnittstelle, welche gemeinsam e Methoden und Funktionen definiert die in Unterschiedlichen Klassen impementiert

	werden.
Iptables	IP Filter

10 J

Begriff	Erklärung
Javascript	Skriptsprache, welche hauptsächlich in Webbrowsern eingesetzt wird

11 K

Begriff	Erklärung
Kernel	Zentraler Bestandteil eines Betriebssystems
KVM	Kernel Based Virtual Machine

12 L

Begriff	Erklärung
Layer 2	Die Sicherungsschicht im OSI-Referenzmodell
Layer 3	Die Vermittlungsschicht im OSI-Referenzmodell
Link	Verweis

13 M

Begriff	Erklärung
MPP	Multi Provider Portal
Multicast	Überträgt eine Nachricht von einem Punkt zu einer Gruppe
MS	Meilenstein

14 N

Begriff	Erklärung
Namespace	Abstrakter Container welcher logische Gruppen, z.B. Klassen, enthält.

15 O

Begriff	Erklärung
Overlay-Image	Gepseicherte Differenzen zum Basisimage

16 P

Begriff	Erklärung
Parsing	Analysiert und wandelt eine Eingabe in ein Format um, welches brauchbar für die Weiterverarbeitung ist
Prototyping	Methode, welche ein frühzeitiges Feedback bezüglich einer Lösung ermöglicht.
Prozess	Ablaufendes Programm

17 Q

Begriff	Erklärung
qcow 2	Format für eine Image

18 R

Begriff	Erklärung
raw	Rohdatenformat
Regex	Regulärer Ausdruck welcher Mengen und Untermengen von Zeichenketten beschreibt
rsyslog	Protokolliert Systemereignisse
RTM	Ready To Manufacture

19 S

Begriff	Erklärung
Service	Ein Mechanismus um Zugriff auf eine oder mehrere Ressourcen zu haben
Server	Programm welches auf Clients wartet , welche für ihn Dienstleistungen erfüllen
Shadow Copies	Entwickelte Technologie von Microsoft um Snapshots zu erstellen der Daten auch wenn diese gelockt sind.
Surfer	Charakter mit verschiedenen Eigenschaften wie er im Internet Surft

String	Zeichenkette
SVN	Subversion
SW	Software
Szenario	Definiert Surfertypen und wie viele von denen gebraucht werden

20 T

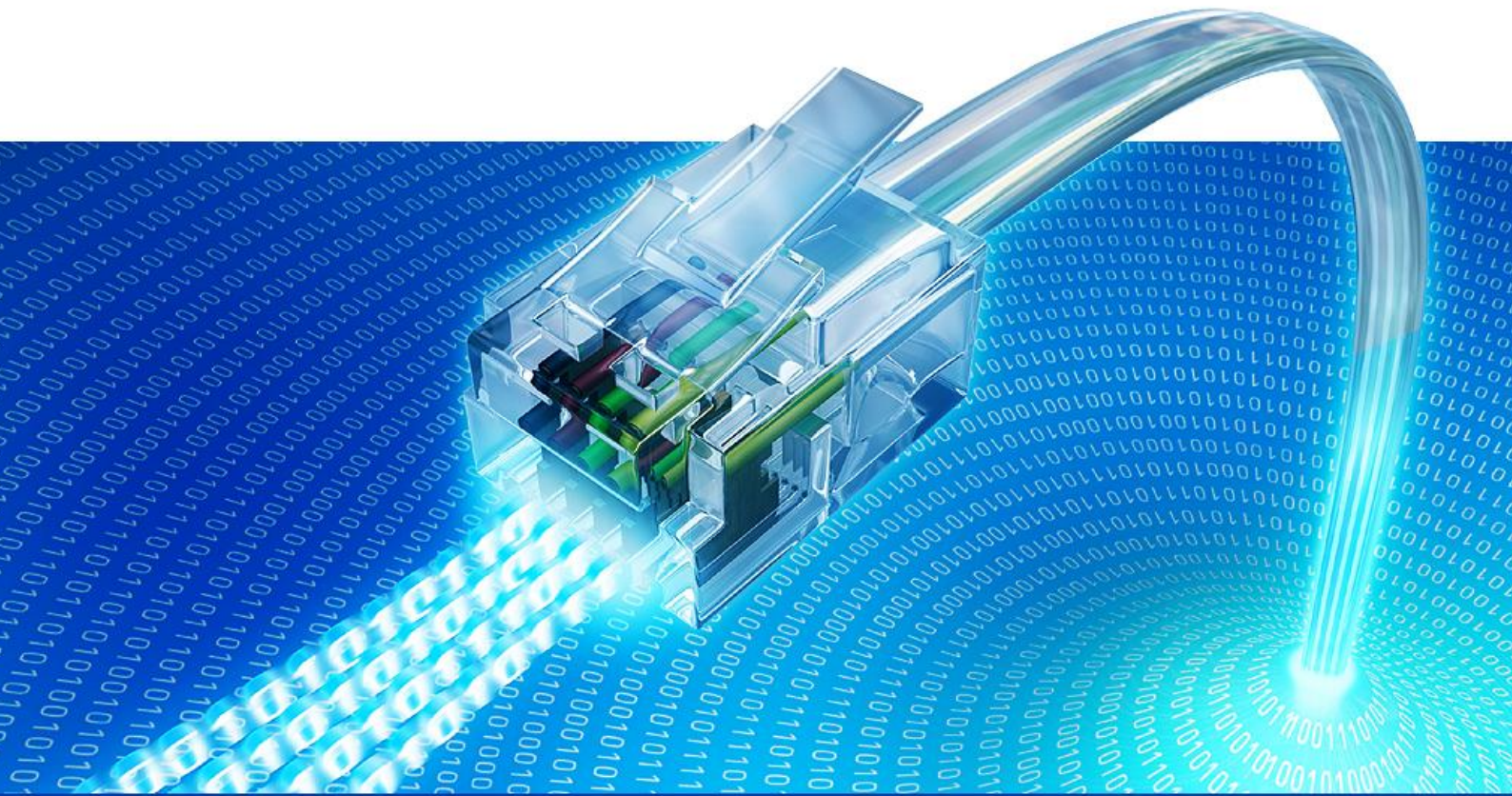
Begriff	Erklärung
Team Seite	Jedes Projekt auf dem Team Foundation Server hat ein Portal resp. Team Seite
TFS	Team Foundation Server
Thread	Teil eines Prozesses
Tweaks	Verbessern, Fine-Tuning and einer Komponente

21 U

Begriff	Erklärung
UML	User Mode Linux
URL/URI	Identifizieren und lokalisieren Ressourcen. Im Falle von URL über das HTTP oder FTP Protokoll

22 V

Begriff	Erklärung
VOIP	Voice over IP
VM	Virtuelle Maschine
VNUML	Virtual Network User Mode Linux



6. Literaturverzeichnis

Captive Portal Load Generator

Semesterarbeit HS 2009

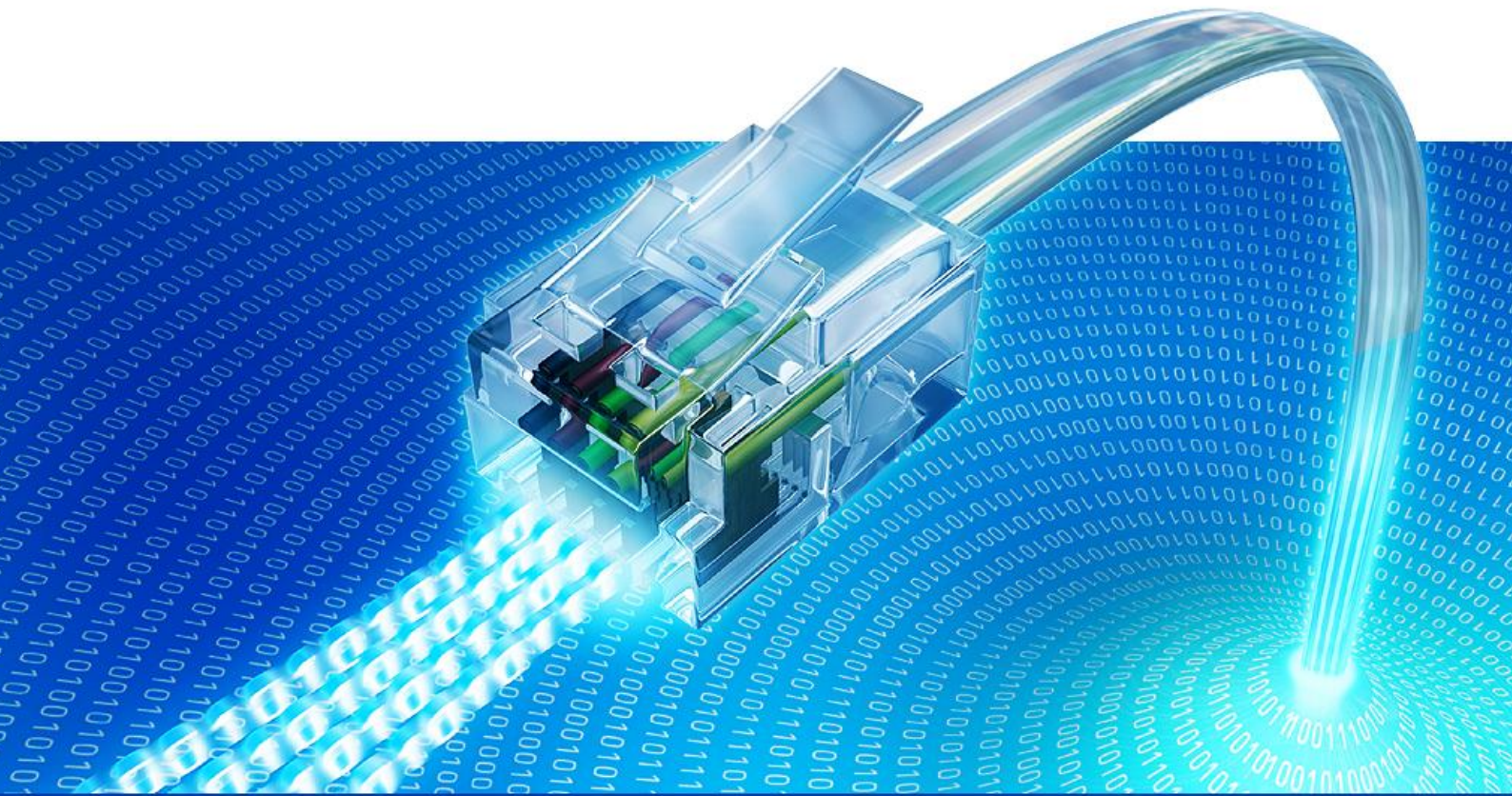
Technische Hochschule Rapperswil

1 **Literaturverzeichnis**

Kühnel, A. (2008). *Visual C# 2008*. Galileo Computing.

Microsoft. (01. 11 2009). Microsoft Developer Network.

Obermeyer, P., & Hawkins, J. (16. 02 2002). Microsoft .NET Remoting: Ein technischer Überblick.



7. Projektplan

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Projektorganisation

Das Team besteht aus 2 einander gleich gestellten Teammitgliedern. Herr Prof. Beat Stettler und Herr Michael Schneider fungieren als Kontrollorgan und unterstützen das Projekt bei offenen Fragen und sind für die Notengebung zuständig.

Das Team trifft sich zu folgenden Zeiten in der HSR:

- Montag: 12:30 – 16:10
- Dienstag: 08:10 – 16:10
- Mittwoch: 12:30 – 15:10
- Freitag: 12:30 – 16:10

1.1 Organisationsstruktur

Alle Teammitglieder erhalten mehrere Aufgabenbereiche zugeteilt. Sie übernehmen dabei die Führungsrolle in diesem Bereich. Wenn Termine nicht eingehalten werden können, oder sonstige Probleme auftreten, sind sie dafür verantwortlich, dies an den Sitzungen zu thematisieren. Die einzelnen Aufgaben innerhalb eines Bereiches werden unter allen Mitgliedern aufgeteilt.

Name	Tätigkeit	Verantwortung
Ymyr Osman	Entwicklung	Projektmanagement, Virtualisierung, User Interface
Oliver Zürcher	Entwicklung	Dokumentation, Analyse, Implementation

1.2 Externe Schnittstellen

Name	Tätigkeit	Verantwortung
Beat Stettler	Beratung	Betreuung und Kontrolle
Michael Schneider	Beratung	Betreuung und Kontrolle

2 Management Abläufe

2.1 Projekt Kostenvoranschlag

Dem Projekt stehen pro Woche und pro Teammitglied ca. 17 Stunden zur Verfügung. Das Arbeitspensum pro Mitglied kann bei Problemen um 2-3 Stunden erhöht werden.

Der Projektstart wurde auf den 14. September und der Abgabe-Termin auf den 18. Dezember festgelegt.

2.2 Zeitplan

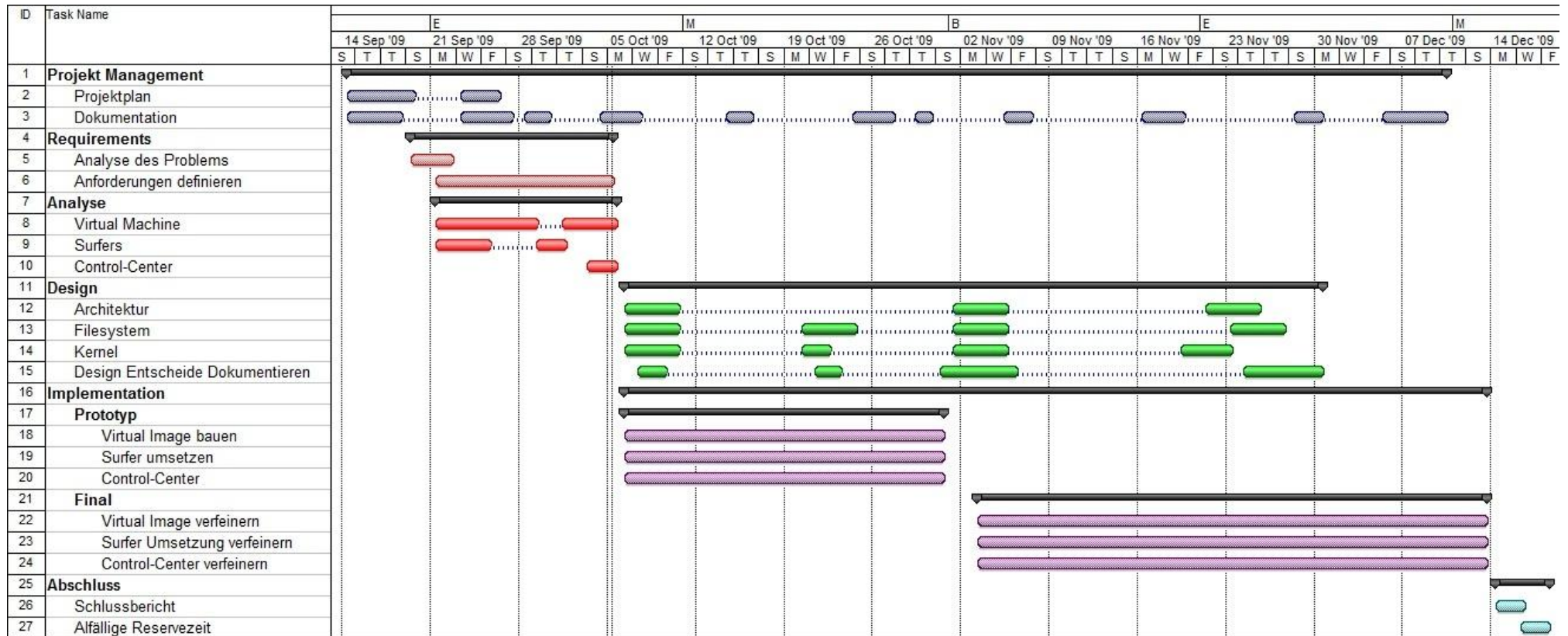


Abbildung 6: Zeitplan

2.3 Meilensteine

Woche	Inhalt	Meilenstein
4	Anforderungen definiert, Problematik analysiert, Arbeitspakete definiert, User Interface Sketch, Anforderungen Spezifiziert, Projektplan	MS1: Anforderungen und Analyse
7	Ein Surfer Implementiert, Starten und Stoppen der Virtuellen Maschinen	MS2: Prototyp CLG
13	Alle Surfer Implementiert, Starten und Stoppen der Virtuellen Maschinen über User Interface, Befehle auf Kommando Ausführen, Statistiken im User Interface, Szenarien festlegen und starten	MS3: RTM CLG
14	Schlussberichte, Abgabe des Projektes	MS4: Projektabgabe

2.4 Besprechungen

Das Team trifft sich mit den Betreuern jeweils dienstags um 09:00 Uhr. Besprechungen werden alle Protokolliert und sind im Anhang zu finden.

2.5 Releases

Release	Woche	Beschreibung
Prototyp	7	Prototyp „Captive Load Generator“. Es können virtuelle Maschinen gestartet und gestoppt werden. Auf dem VMs startet automatisch ein Surfer.
RTM	13	Auslieferbare Version des Captive Load Generator. Es können Szenarien über das User Interface gewählt und angepasst werden, um die Simulation zu starten. Zusätzlich werden alle Interaktionen der virtuellen Maschinen statistisch ausgewertet und im User Interface gezeigt.

2.6 Artefakte

Meilenstein	Artefakte
MS1	Semesterarbeit.pdf
MS2	Prototyp "Captive Load Generator" Semesterarbeit.pdf
MS3	„Ready to Manufacture“ Release des Captive Load Generator
MS4	Semesterarbeit.pdf Schlussbericht_YmyrOsman.pdf Schlussbericht_OliverZürcher.pdf

3 Risiko Management

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	gewichteter Schaden [h]	Massnahmen zur Vermeidung / Verhinderung	Vorgehen bei Eintreffen
R1	Ausfall Arbeitsstation	HW eines Projektmitglieds fällt aus	4	5%	0.2	- Aktuelle Hardware verwenden (Aktueller Zustand der Hardware überprüfen und Mängel, sofern möglich, beheben) - Aktuelle Software verwenden (Updates, Virenschutz)	- Arbeitsplatz Rechner der HSR verwenden - Eigener PC zuhause verwenden
R2	Ausfall Netzwerkinfrastruktur	Netzwerk an der HSR oder Anbindung des MPPs fällt aus	48	5%	2.4	- Datentransfermöglichkeiten innerhalb des Teams bereithalten: USB, CD-Rom	- Daten über anderes Netzwerk oder Medien austauschen
R3	Datenverlust	Datenverlust durch HW oder SW Fehler	480	1%	4.8	- Lokale Kopie der Daten - Daily Backup auf externe redundante Storage - SVN / TFS	-Daten aus dem Backup wiederherstellen - Wenn Daten verloren sind, abklären ob das Projekt gefährdet ist
R4	Ausfall eines Projektmitglieds	Ein Projektmitglied erleidet einen Unfall oder wird krank	240	2.5%	6	-Verantwortlichkeit im Team definieren	- Arbeit innerhalb des Teams aufteilen
R5	Streit im	Es entstehen	24	5%	1.5	- Konstruktive Kritik und	- Ursachen früh

	Projektteam	Streitigkeiten im Projektteam				Kommunikation im Projektteam pflegen und fördern	Erkennen und vermeiden
R6	Fehleinschätzungen des Aufwandes	Zeitplan wird nicht eingehalten weil der Aufwand falsch eingeschätzt wurde	30	10%	3	-Kritische Arbeitspakete frühzeitig bearbeiten -Wöchentliche Sitzung mit Betreuern	- Aussortieren gewisser Funktionen und Features in den Sitzungen
R7	Fehleinschätzung durch Annahmen	Falsche Implementierungen durch Annahmen	15	5%	0.75	-Sobald eine Annahme getroffen wird mit Teammitglied absprechen / nachfragen	- Teamdiskussion einberufen
R8	Probleme mit Technologien	Einarbeitung in Linux, Virtualisierungsmöglichkeiten	30	10%	3	- Dokumentation und Anleitungen im Internet und Sachbüchern suchen	- Mehr Zeit in Einarbeitung investieren. Projektplan muss eingehalten werden
R9	Probleme bei der Kommunikation von Hosts mit VMs	Befehle können nicht an die virtuellen Maschinen gerichtet werden	30	10%	3	- Verschiedene Mechanismen für eine Kommunikation in Betracht ziehen und studieren	- Kommunikationslösung austauschen durch eine andere Möglichkeit
Sum			921	56%	24.65		

Unsere Zeitplanung enthält eine Reserve von ca. 35 Stunden, was 17.5 Stunden pro Teammitglied entspricht. Die Risiko Analyse zeigt, dass wir ein Risiko von 24.65 Stunden haben. Unsere Zeitplanung enthält daher eine Reserve von 35 Stunden. Durch diese Reserve können wir den gewichteten Schaden abdecken.

4 Arbeitspakete

Nr	Name	Inhalt / Arbeitsresultat	Soll [h]	Ist[h]
1	Projekt Management			
1.1	Projektplan	Zeigt die Einteilung der Einzelnen Pakete in den 14 Wochen	14	14
1.2	Dokumentation	Ausarbeitung der Vorlagen und dokumentieren des Projektes während der gesamten Projektdauer.	120	132.5
2	Requirements			
2.1	Analyse des Problems	Analyse der Anforderungen um eine Übersicht zu erarbeiten	5	5
2.2	Anforderungen definieren	Anforderungen und Ziele an das Projekt definieren und festlegen	10	10
3	Analyse			
3.1	Virtuelle Maschinen	In diesem Arbeitspaket wird die Idee erarbeitet, wie die Virtualisierung umgesetzt werden soll.	75	71.5
3.2	Surfers	Mit Surfer ist das Programm gemeint, welches auf den virtualisierten Clients läuft, um Last zu generieren. Ziel dieses Paketes ist es, verschiedene Surfcharakteristiken zu formulieren, welche später umgesetzt werden. Zudem soll noch die Frage geklärt werden, ob eine Programmier – oder Skriptingsprache zur Entwicklung verwendet wird.	5	5
3.3	Control-Center	Überlegungen rund um das Control-Center. Was die Ziele sind und welche Funktionalitäten es zur Verfügung stellen soll. GUI Sketches sind dabei ein wichtiges Hilfsmittel.	5	5
4	Design			
4.1	Architektur	Definition der Architektur, d.h. wie die Clients gesteuert werden. Zum Beispiel	4	14

		ob es nun von einem zentralen Punkt aus geschieht oder nicht		
4.2	Dateisystem	Das Dateisystem, welches von einer virtuellen Maschine verwendet wird. Im Laufe des Projektes wird dies optimiert, indem nicht verwendete Elemente entfernt werden. Reduzierung auf ein Minimum.	20	9.5
4.3	Kernel	Konfiguration des verwendeten Kernels.	8	3.5
4.4	Design-Entscheidung Dokumentieren	Dokumentieren der Design-Entscheidung	8	12
5	Implementation			
5.1	VM Image erstellen	Basis VM-Konfiguration erstellen, welche beliebig mal für die Szenarien geklont werden kann.	15	15
5.2	Surfers Implementieren	Die erarbeiteten Surfer aus der Analyse implementieren.	30	30
5.3	VMRemoteControl implementieren	VMRemoteControl implementieren	35	35
5.4	Control-Center	Das Kontrollcenter ist der Dreh und Angelpunkt für den Benutzer. Es besteht aus folgenden Teilpunkten: Steuerung der Surfer (Szenario starten/stoppen) Nimmt Meldungen von Surfer entgegen Zeigt Statistiken an Verwaltung von Szenarios	90	90
6	Abschluss			
6.1	Schlussbericht	Erstellung des Schlussberichtes	4	4
6.2	Reservezeit	Puffer von 3 Tagen bei allfälligen Problemen		
7	Sitzungen			
7.1	Sitzungen mit Betreuern	Team sitzt mit Betreuern zusammen und bespricht Fortschritt und weiteres Vorgehen. Die Sitzung ist jeweils dienstags um 09:00	14	14

5 Infrastruktur

5.1 Räumlichkeiten

Besprechungen finden im Büro 6.001 der HSR statt. Hauptsächlich wird im Studienarbeitszimmer 1.258 gearbeitet. Bei Bedarf kann auch von zu Hause aus gearbeitet werden.

5.2 Hardware

Primär arbeitet jeder mit seinem Studienarbeits-PC. Bei Ausfall sind persönliche Notebooks vorhanden. Als Versionsverwaltung dient uns ein Team Foundation Server.

5.3 Software

- **Betriebssysteme**
 - Microsoft Windows XP
 - Microsoft Windows 7
 - Debian Lenny
 - Ubuntu
- **Programmiersprache**
 - Microsoft C#.NET / Mono
- **Entwicklungsumgebung**
 - Microsoft Visual Studio Team System 2008
 - Monodevelop 2.0
- **Versionsverwaltungssoftware**
 - Microsoft Team Foundation Server 2008 / MSSQL 2008
- **Dokumentation**
 - Microsoft Word 2007
 - Microsoft Project 2007
 - Microsoft Excel 2007

5.4 Backup

Der Team Foundation Server läuft auf einem Microsoft Windows 2008 Enterprise Server. Auf diesem Server werden zweimal täglich „shadow copies“ für kleinere Probleme erstellt. Des Weiteren wird der komplette Server jeweils in der Nacht von Samstag auf Sonntag voll und in jeder Nacht inkrementiell gesichert. Backups werden für 2 Wochen gespeichert.

5.5 Kommunikation

Der Team Foundation Server unterstützt uns bei der Kommunikation. Es existiert eine Team Seite (Sharepoint Portal) in der alle Dokumente verwaltet werden können. Es werden alle Artefakte „versioniert“, nicht nur der Sourcecode.

Zusätzlich unterstützen uns folgende Kommunikationsmittel für einen erfolgreichen Projektverlauf:

- **Besprechungen**

- E-Mail
- MSN-Messenger
- Teamspeak (VOIP)

6 Qualitätsmassnahmen

6.1 Dokumentation

Grosser Wert wird darauf gelegt, dass die Dokumentation stets aktuell und vollständig ist. Änderungen müssen möglichst schnell von den Teammitgliedern eingesehen werden. Falls es zu Konflikten kommt, welche nicht ohne weiteres zusammengeführt werden können, muss das Dokument mit einem Suffix im Format<DOKUMENT_NAME>_<NAME> auf der Team Seite genannt werden. Diese Konflikte werden dann im Team besprochen und behoben.

6.2 Besprechungsprotokolle

Sitzungen werden jeweils von einem Teammitglied protokolliert. Mit dieser Massnahme möchten wir Besprochenes schriftlich festhalten (Kritik, Vorschläge, Anregungen usw.).

6.3 Arbeitszeit

Alle Projektmitarbeiter erfassen ihre geleistete Arbeit im Dokument Zeiterfassung.xlsx. Jedes Teammitglied ist verantwortlich, jeweils Ende Woche eine aktualisierte Version auf der Team Seite zu publizieren.

6.4 Versionskontrolle

Durch den Einsatz eines Team Foundation Servers, ist die Versionskontrolle direkt in den von uns eingesetzten Programmen integriert (Word, Excel, Visual Studio, Explorer usw.).

7 Projektauswertung

7.1 Zeitauswertung

Eine gute Planung in einem Projekt ist wichtig und massgebend für den Erfolg des Projektes. Der Zeitplan ist hierbei Indikator um die Planung zu messen. Er kann jedoch nicht immer zu 100% übereinstimmen mit dem schlussendlichen Ist-Wert. Es gibt immer Aufgaben bei denen mehr oder weniger Zeit gebraucht wird. Ziel ist es den Zeitplan im Grossen und Ganzen einzuhalten.

In diesem Projekt wurde dieses Ziel erreicht. Die Zeitplanung stimmte mehrheitlich mit den Ist-Werten überein. Geplant waren 480 Stunden, um das Projekt im Team umzusetzen. Gebraucht wurden allerdings 502.5 Stunden. Die 22.5 Stunden mehr entsprechen ca. einem halben Tag Mehraufwand je Teammitglied. Grosse Unterschiede entstanden in zwei Aufgabenbereichen. Diese sind das Dateisystem aufzubauen und die Architektur. Bei Erstem wurde zu viel Zeit eingeplant, da noch nicht entschieden war, welche Virtualisierungsmöglichkeit gewählt wird. Die Architektur ergab mehr Aufwand, weil während des Projektes eine grundlegende Änderung getroffen wurde.

7.1.1 Zeitauswertung pro Iteration

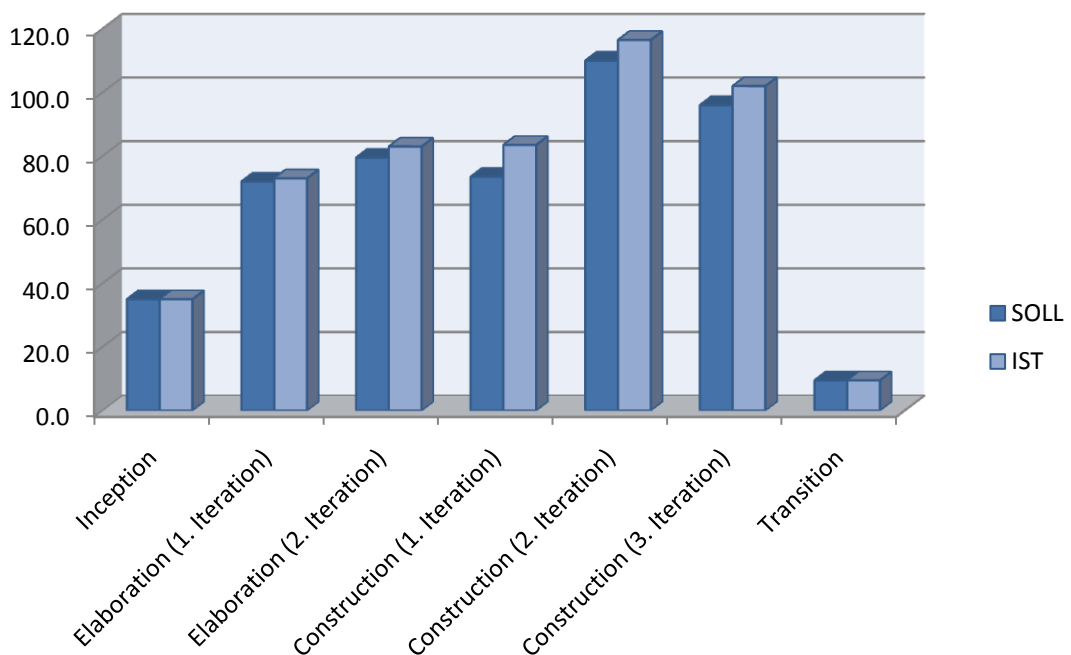


Abbildung 7: Zeitauswertung pro Iteration

7.1.2 Zeitauswertung pro Disziplin

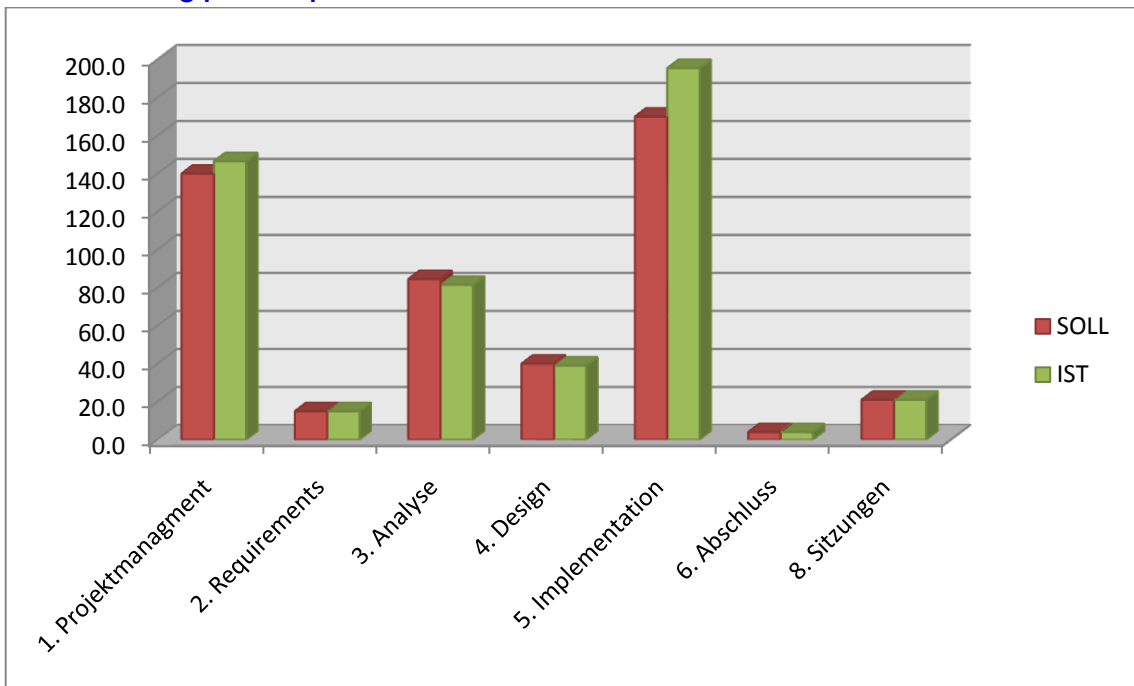


Abbildung 8: Zeitauswertung pro Disziplin

7.1.3 Zeitauswertung pro Semesterwoche

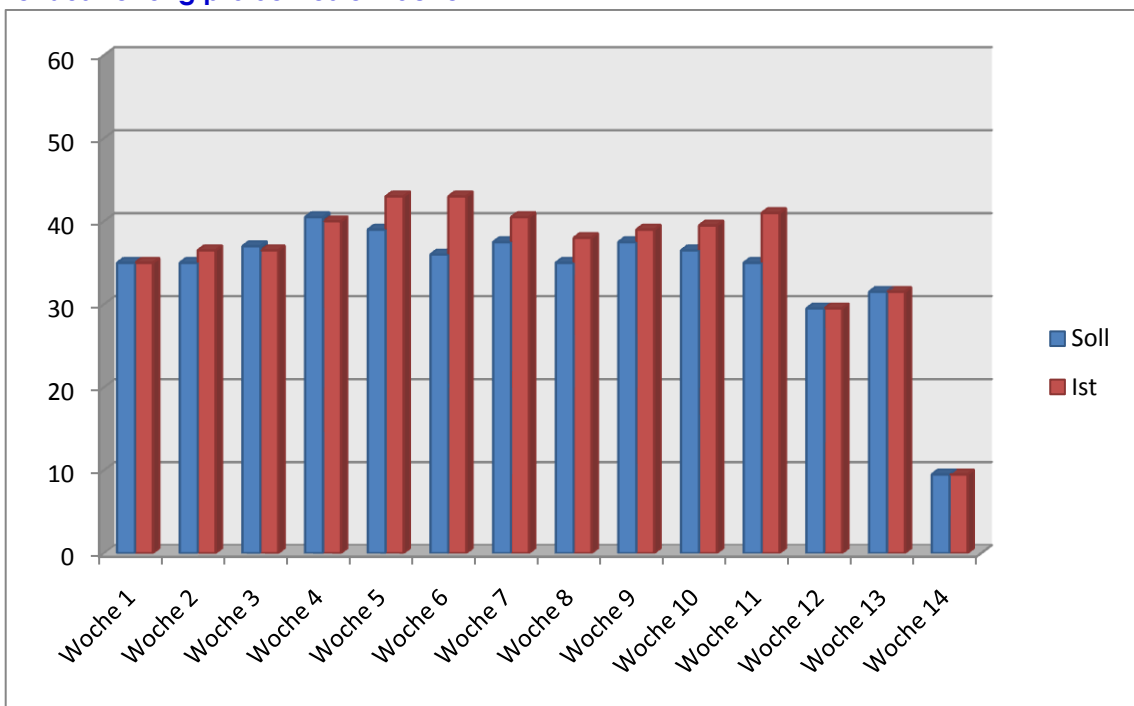


Abbildung 9: Zeitauswertung pro Semesterwoche

7.2 Codeauswertung

Um sich den Umfang des Projektes besser vorstellen zu können, zeigen die folgenden Abbildungen die Anzahl Zeilen im Quellcode. Hierbei wurden zwei verschiedene Varianten gebraucht. Erstens sind die Anzahl Zeilen mit einem herkömmlichen Tool gezählt. Als zweite Variante bietet das Visual Studio in der Team System Version eine Code Analyse an. Bevor erklärt wird was die Visual Studio Code Analyse anders macht, wird Bill Gates zitiert um, die Motivation zu erklären, warum es eine extra Analyse von Microsoft gibt.

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.”

Die Code Analyse von Visual Studio beinhaltet folgende zusätzliche Punkte:

Maintainability Index

Indikator von 0 bis 100 welcher die Wartbarkeit der enthaltenen Typen misst. Dabei gilt 0 als schlecht Wart bar und 100 als hoch Wart bar

Cyclomatic Complexity

Dieser Wert misst auf jeder Ebene die gesamte Anzahl der individuellen Wege im Code. Dieser berechnet sich durch Entscheidungspunkte. Das heisst Anweisungen, welche z.B. if-Blöcke, switch-Anweisungen oder Schleifen beinhalten. Hier gilt ein tieferer Wert als besser.

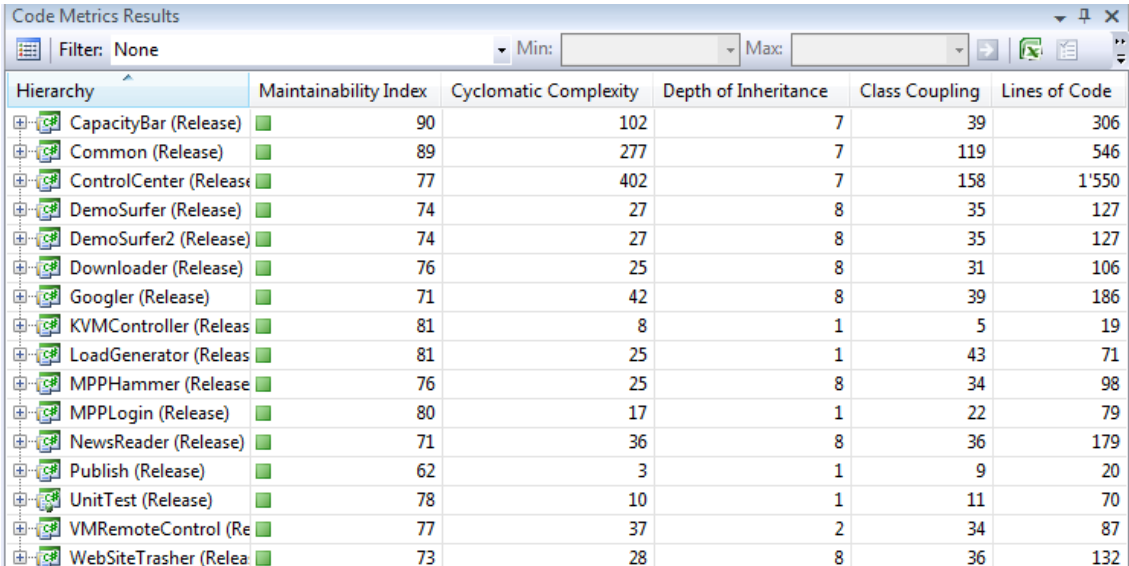
Depth of Inheritance

Dieser Wert zeigt zeigt, wie viele Typen sich über dem aktuellem Typen in der Vererbungshierarchie befinden. Eine hohe Vererbung kann ein Anzeichen von Over-engineering sein. Das heisst, die Komplexität des Testens oder die Wartbarkeit der Applikation steigt.

Class Coupling

Dieser Wert zeigt die gesamte Anzahl Abhängigkeiten eines Typs. Dabei wird geachtet, dass die primitiven Datentypen wie z.B. Integer oder String nicht mitgezählt werden. Ein tiefer Wert heisst hierbei, dass dieser Typ wiederverwendet werden könnte.

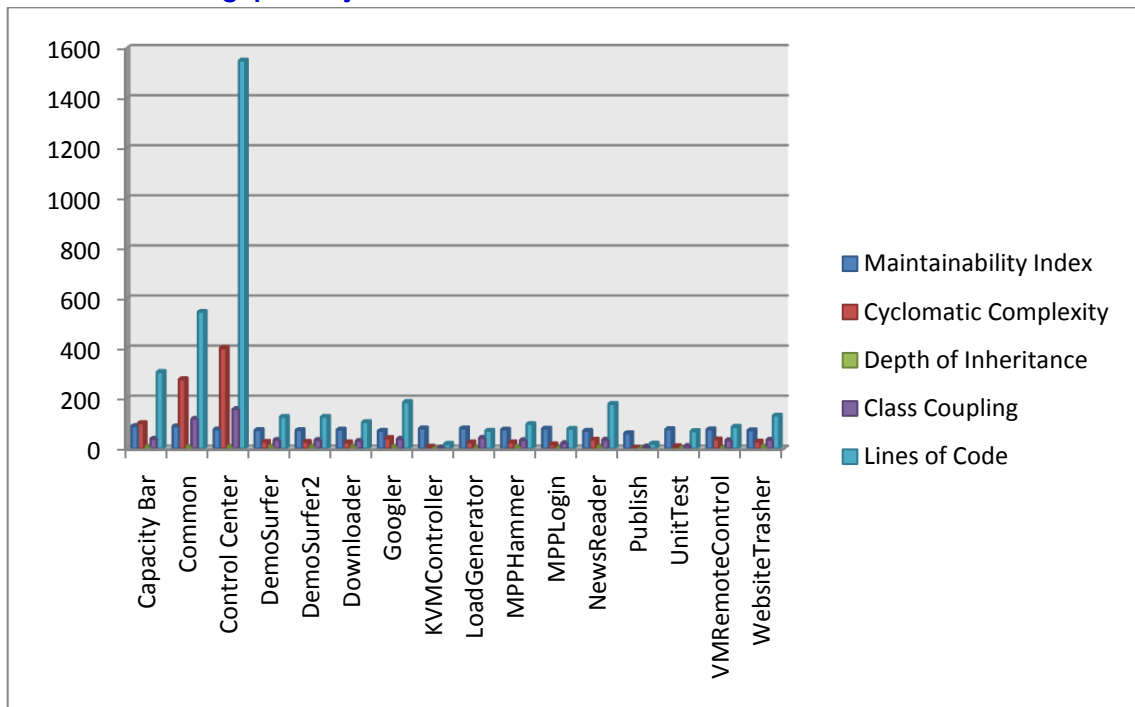
7.2.1 Visual Studio Code-Auswertung



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
CapacityBar (Release)	90	102	7	39	306
Common (Release)	89	277	7	119	546
ControlCenter (Release)	77	402	7	158	1'550
DemoSurfer (Release)	74	27	8	35	127
DemoSurfer2 (Release)	74	27	8	35	127
Downloader (Release)	76	25	8	31	106
Googler (Release)	71	42	8	39	186
KVMController (Release)	81	8	1	5	19
LoadGenerator (Release)	81	25	1	43	71
MPPHammer (Release)	76	25	8	34	98
MPPLogin (Release)	80	17	1	22	79
NewsReader (Release)	71	36	8	36	179
Publish (Release)	62	3	1	9	20
UnitTest (Release)	78	10	1	11	70
VMRemoteControl (Release)	77	37	2	34	87
WebSiteTrasher (Release)	73	28	8	36	132

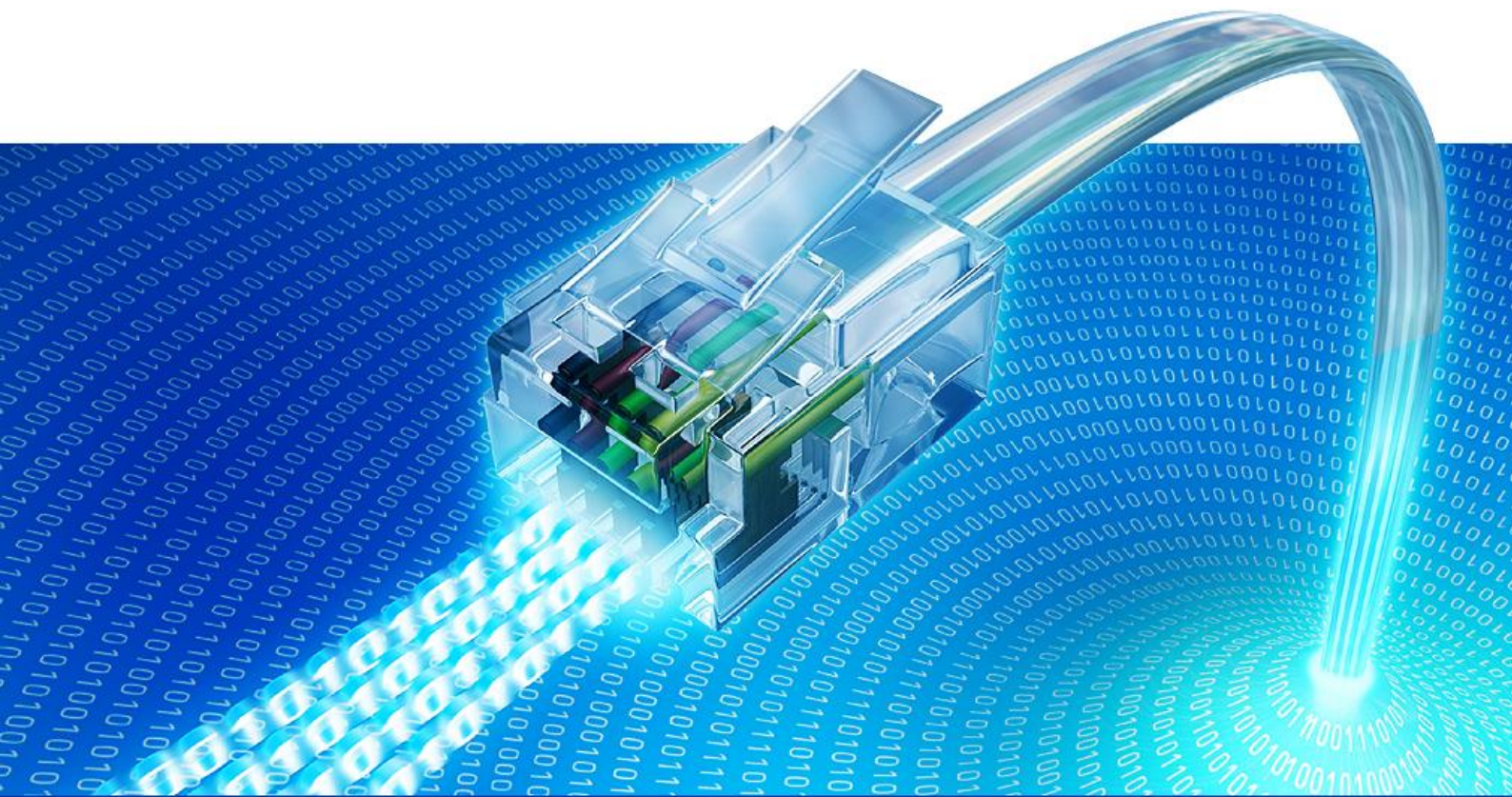
Abbildung 10: Codeauswertung von Visual Studio

7.2.2 Code-Auswertung pro Projekt



7.2.3 Code-Auswertung gesamtes Projekt

	Gesamt/Summe
Maintainability Index	76.88
Cyclomatic Complexity	68.19
Depth of Inheritance	5.25
Class Coupling	48.12
Lines of Code	3703



8. Anforderungsspezifikationen

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Allgemein

1.1 Produktperspektive

Durch den grossen Anstieg an Clients wurde das Testen des MPPs immer komplizierter. Das Produkt soll Abhilfe schaffen indem es die Virtualisierung automatisiert.

1.2 Produkt Funktion

Folgende Funktionen werden gefordert:

Konfiguration eines Szenarios

Zur Konfiguration eines Szenario gehört die Festlegung der Anzahl Clients, deren Surfverhalten und die Anzahl an Durchgängen, welche auch unendlich lang sein kann.

Starten/Stoppen eines Szenarios

Ein Szenario kann von einem Computer aus gestartet werden. Die Clients werden auf alle verfügbaren Nodes automatisch aufgeteilt.

Konfiguration des VM-Images

Das vorhandene Linux Image, welches als Datei zur Verfügung steht und für alle Clients verwendet wird, kann auch zu späteren Zeitpunkten auf die gewünschte Weise angepasst werden. So sollten auch bei Softwareaktualisierungen keine Probleme auftreten. Das Image ist ein wichtiger Faktor für die Performance/Skalierbarkeit. Hier kann viel optimiert werden (Kernel Tweaks, möglichst wenig Software installieren usw.).

1.3 Benutzercharakteristik

Die Applikation richtet sich an erfahrene Benutzer, welche mit dem MPP vertraut sind.

1.4 Einschränkungen

Keine

1.5 Annahmen

Keine

1.6 Abhängigkeiten

Die Applikation ist vom MPP abhängig. Wenn die Landing-Page zu sehr abgeändert wird, muss eventuell Code angepasst werden.

2 Spezifische Anforderungen

Die Anforderungen werden nach dem FURPS+-Modell kategorisiert. Diese Qualitätsmerkmale dienen als Checkliste für die Behandlung von Anforderungen, um das Risiko zu verringern, eine wichtige Facette des Systems zu übersehen.

2.1 Funktionalität

2.1.1 Erweiterbarkeit

Das Surfverhalten ändert sich oft. Deshalb soll es möglich sein, zu einem späteren Zeitpunkt weitere Surfertypen zu implementieren. Diese sollen ohne Anpassung im Code der Applikation hinzugefügt werden können.

2.2 Bedienbarkeit

2.2.1 Erlernbarkeit

Die Einarbeitungszeit in die Applikation soll lediglich wenige Minuten in Anspruch nehmen. GUI Elemente sind intuitiv zu platzieren und zu beschriften.

2.2.2 Wiederherstellbarkeit

Bei einem Programmabsturz dürfen die Laufzeitdaten des aktuellen Szenarios nicht verloren gehen. Einstellungen sollen ebenfalls erhalten bleiben.

2.2.3 Fehlerbehandlung

Die Anwendung soll den Benutzer vor Fehler schützen. Der Benutzer soll von der Konfiguration bis zum Ende eines Szenarios klar geführt werden.

2.3 Zuverlässigkeit

Abstürze dürfen durch die Applikation selbst keine entstehen. Externe Faktoren, welche die Applikation zum Abstürzen bringen, können natürlich nicht ausgeschlossen werden.

2.4 Leistung

Es gilt die Ressourcen möglichst optimal einzusetzen. Da mehrere tausend VMs in ein Szenario involviert werden können, ist die Skalierbarkeit ein wichtiger Faktor. Wenn mehr Ressourcen benötigt werden, soll das System um weitere Rechner erweitert werden können.

3 Wartbarkeit

Der Code soll gut strukturiert abgelegt werden. Dabei gehören logische Gruppen von Klassen und Typen in gemeinsame Namespaces. Zirkuläre Abhängigkeiten sind verboten.

4 Schnittsteellllen

4.1 Benutzerschnittstelle

Die Applikation ist Grundsätzlich über ein GUI zu steuern.

4.2 Softwareschnittstelle

- Programmiersprache
- Microsoft C#.NET 3.5

- GUI Framework
- WinForms
- MPP
- Zugriff via http

4.3 Lizenzanforderungen

Das Endprodukt ist ein lauffähiger Prototyp. Das INS kann über diesen Prototypen frei verfügen.

4.4 Verwendete Standards

Im gesamten Projekt wird nach den normalen Microsoft Coderichtlinien programmiert.

4.5 Sicherheit

Keine speziellen Anforderungen.

5 Use Cases

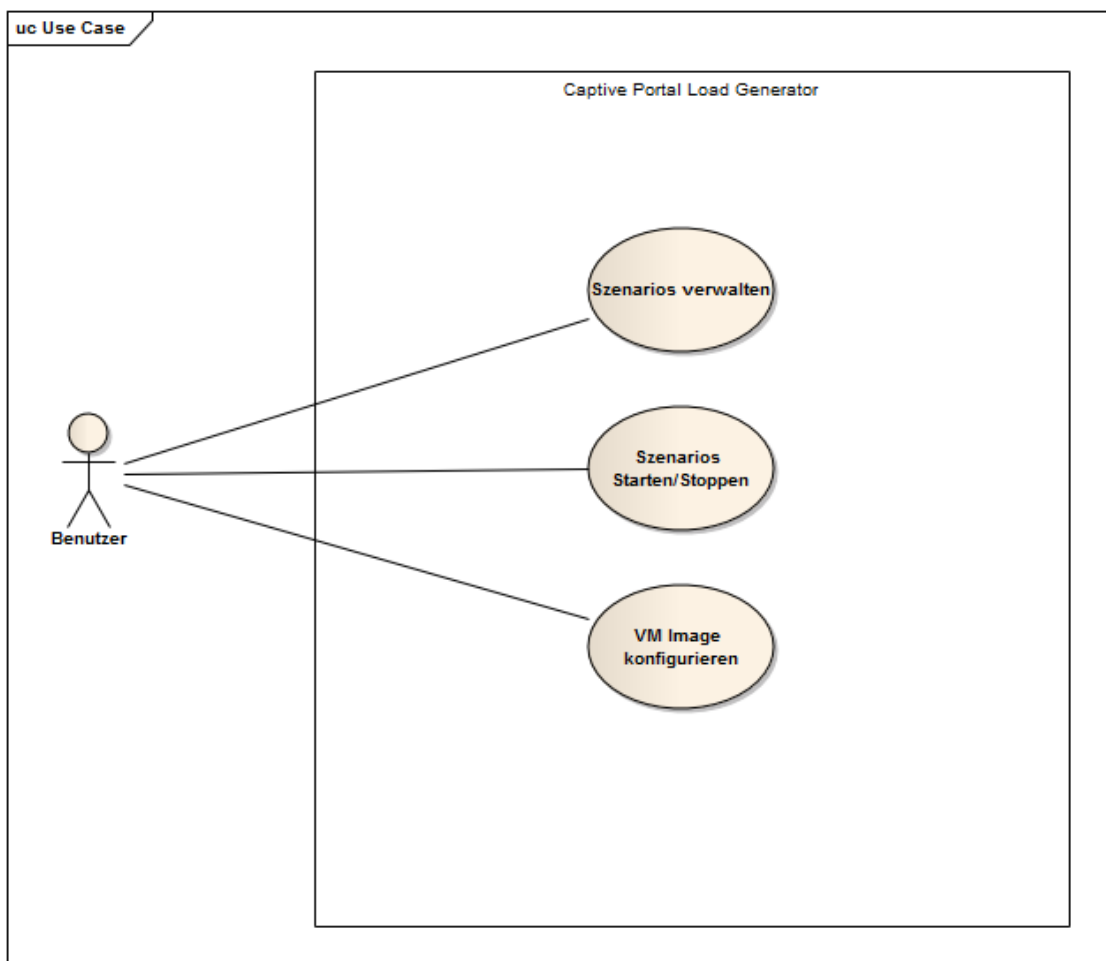


Abbildung 11: Use Cases

5.1 UC 1: Szenarios verwalten

In einem GUI kann der Benutzer Szenarien verwalten. Zu einem Szenario gehören verschiedene Surfertypen. Pro Surfertyp können folgende Eigenschaften verändert werden:

- Anzahl Instanzen
- Anzahl Durchläufe pro Instanz
- Für den Typen spezifische Einstellungen in einem Subdialog

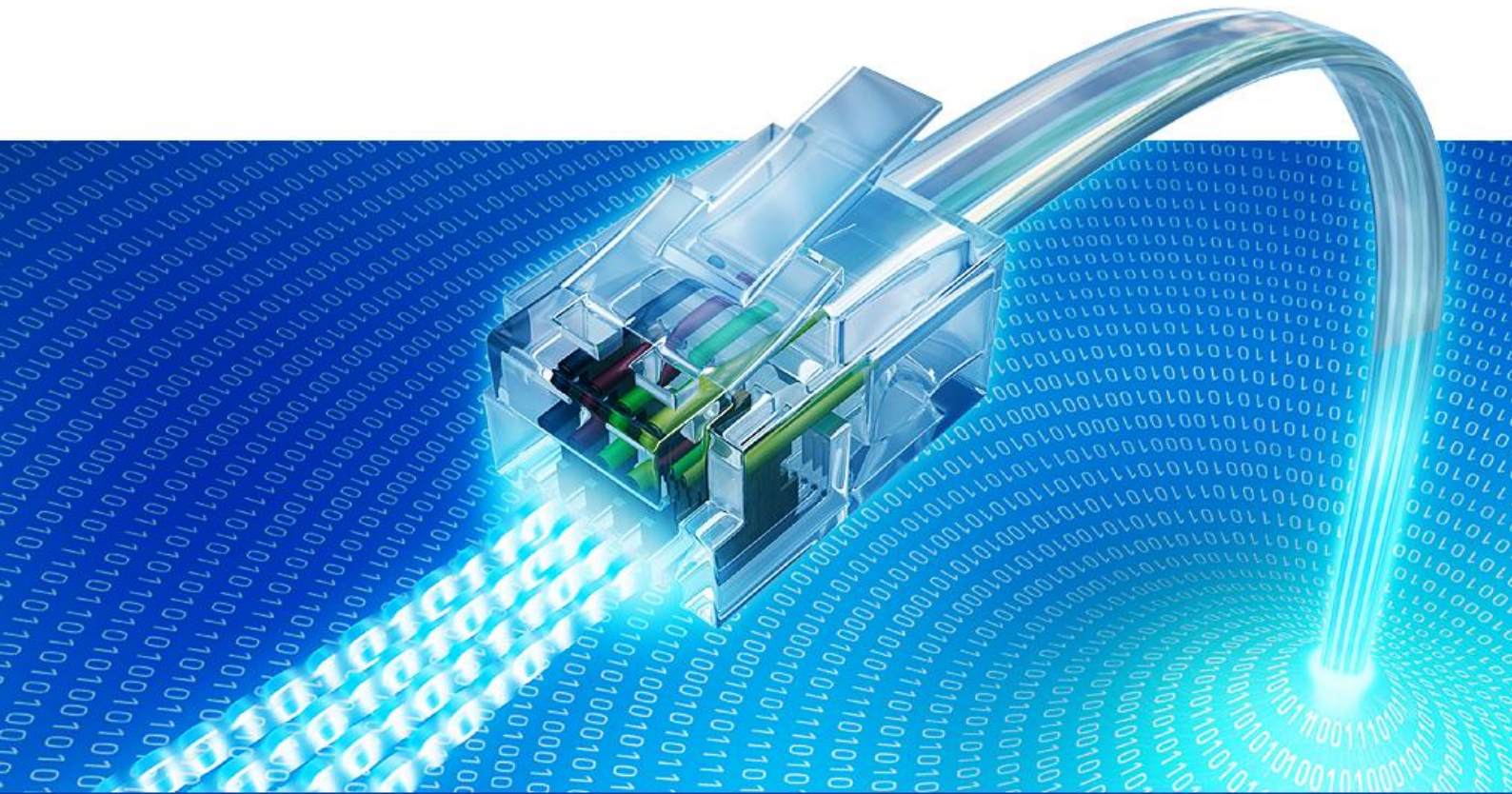
Die Szenarien können über ein „speichern“ Knopf gespeichert werden.

5.2 UC 2: Szenarios Starten/Stoppen

Über einen Start Knopf kann das Szenario gestartet werden. Alle VMs werden hochgefahren. Der Fortschritt wird dem Benutzer pro VM angezeigt. Wenn alle VMs aufgestartet sind, kann über einen weiteren Startknopf das Multicast Signal an alle VMs geschickt werden, damit diese beginnen, Last zu generieren.

5.3 UC 3: VM Image konfigurieren

Das VM Image, welches von jedem Client verwendet wird, kann gebootet werden. Alle Einstellungen müssen in dem Image gespeichert werden. Das neue Image muss anschliessend auf alle Nodes neu verteilt werden.



9. Analyse & Design

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Analyse Virtualisierungssoftware

2000 Clients zu simulieren, stellt Anforderungen an die Hardware. Je nach Virtualisierungssoftware können diese Anforderungen voneinander abweichen. In diesem Kapitel wird untersucht, wie viele Ressourcen pro VM für die verschiedenen Virtualisierungsprogramme benötigt werden. Mit dem Ergebnis können Hochrechnungen gemacht werden, um die Anforderung an die Hardware für ein bestimmtes Szenario abzuschätzen.

1.1 Anforderungen

- Ressourcenverbrauch
- Nur das nötigste im Kernel
- RAM-Belastung möglichst klein
- Dateisystem möglichst klein
- Klonbare Abbilder (Ressourcen sparen)
- Ausführungsgeschwindigkeit
- CPU
- Datendurchsatz im RAM
- Verwaltbarkeit (CLI oder API)
- Existiert eine CLI?
- Ist eine API für Entwickler vorhanden?
- Netzwerk
- Wirtssystem muss mit den VMs kommunizieren können

1.2 Testumgebung / Testdefinition

Für die Tests wird Debian oder Windows XP als Wirtssystem eingesetzt. Die Laborrechner verfügen über 3 GB RAM und ein Intel Core2Duo E6750 Prozessor. Gemessen werden die benötigten Ressourcen pro VM in 2 Testfällen: Wenn eine VM läuft (Test A) und wenn 50 parallel laufen (Test B). Als Gastbetriebssystem wird eine Minimalinstallation von Debian verwendet. Der von uns kompilierte Kernel ist 920 KB gross.

1.3 VMWare

VMWare ist die bekannteste Virtualisierungssoftware auf dem Markt. Es ist für Linux, Mac und Windows verfügbar. Zum ganzen Softwarepaket gehören viele GUIs zum administrieren der virtuellen Maschinen. Für das kostenlose Einstiegsprodukt VMWare Server ist eine Registration notwendig.



1.3.1 Grundlegendes

Für das eingesetzte Debian wurde ein optimierter Kernel kompiliert. Nach der Minimalinstallation wurden die Services cron, rsyslogd und exim4 deinstalliert.

VMWare stellt dem Endbenutzer eine CLI zur Verfügung . Über die CLI sind viele Grundfunktionen wie Starten, Stoppen, Klonen usw. möglich:

Windows: <Installationsverzeichnis>/vmrun.exe
Linux: /usr/bin/vmrun

Eine weitere wichtige Funktion ist die Möglichkeit, Befehle an eine virtuelle Maschine zu senden.

In VMWare können Teams erstellt werden. Dies sind Gruppierungen von virtuellen Maschinen. Die komplette Gruppe kann über den Namen aufgestartet/gestoppt werden. Dabei ist es möglich, einen Intervall festzulegen, in welchem die VMs aufgestartet werden. Das Teamkonzept wäre optimal für unsere Anforderungen an die Verwaltbarkeit. Allerdings ist die Umsetzung etwas knapp: Nur ganz wenige Kommandos wie Start/Stopp können an ein Team gesendet werden. Der Befehl um ein Programm auf einer VM zu starten, muss an eine einzelne Maschine geschickt werden. Des Weiteren kann ein Team nicht über das CLI gebildet werden: Handarbeit über das GUI mit mindestens 3 Mausklicks pro VM, welche hinzugefügt werden sollen, wird benötigt. Dies stellt natürlich ein Problem dar, wenn 2000 VMs zu einem Team hinzugefügt werden sollen.

Die erwähnten Negativpunkte fallen nicht allzu stark ins Gewicht, da die fehlende Funktionalität mit relativ wenig Aufwand selbst programmiert und im Control Center integriert werden kann.

Für dieses Projekt wäre VMWare Workstation oder VMWare Server denkbar. VMWare Server ist gratis, jedoch muss jede einzelne VM, welche gestartet werden soll, in die Storage per Webinterface abgelegt werden. Dies hat dieselben Nachteile, wie die Teamfunktion in der Workstation, kann aber diesmal nicht umgangen werden. 2000 VMs in eine Storage einzutragen, ist extrem aufwändig. In der Workstation hingegen sind VMs direkt über den Pfad zur .vpx-Datei startbar, ohne zuvor in die Storage aufzunehmen.

VMWare Tools wird bei den Tests nicht verwendet, da keine relevanten Tools in diesem Paket enthalten sind.

1.3.2 Test A: 1 VM

Das Betriebssystem belegt nach dem Bootvorgang laut `top` Kommando weniger als 20 MB RAM. Trotzdem benötigt der Prozess, welcher von VMWare gestartet wurde, 53 MB im Wirtssystem.

Eigenschaft	Wert
Grösse des Kernels	0.92 MB
Dateisystem	Ca. 600 MB
RAM	53 MB
Benötigte Zeit zum Aufstarten	35s

1.3.3 Test B: 50 VMs

VMs können in VMWare mit der Option *linked* geklont werden. Klone können sich Ressourcen teilen. Somit muss das Dateisystem nicht für jede VM komplett kopiert werden.

VMWare scheiterte beim Versuch, 50 VMs aufzustarten. Nach der 12. VM meldete VMWare, dass zu wenig RAM vorhanden ist, obwohl noch mehr als 1.6 GB frei war. Deshalb beziehen sich die Testresultate auf ein geändertes Testszenario mit lediglich 10 VMs parallel.

Eigenschaft	Wert
-------------	------

Grösse des Kernels	0.92 MB
Dateisystem	ca 600 MB + ~13MB pro VM
RAM	53 MB pro VM
Benötigte Zeit zum Aufstarten	135s

1.4 VNUML

VNUML baut auf UML (user mode linux) auf.

UML erlaubt dem Benutzer, einen kompletten

Kernel als Anwendungsprozess laufen zu lassen.

Dies kann zu diversen Zwecken als Vorteil genutzt

werden. Zum Beispiel lassen sich ganze

Netzwerkdienste auf einem Rechner getrennt

voneinander laufen. Wenn einer abstürzt, hat dies keinen Einfluss auf den anderen.



VNUML wurde zum Testen von Ipv6 Szenarien im Zusammenspiel mit Linux und der

Zebra/Quagga Routing Suite an der Technischen Universität in Madrid entwickelt. Es ist ein

Parser, welcher XML Dateien mit Netzwerkstrukturen einliest. Dieser startet für alle

konfigurierten Hosts in der XML Datei ein UML auf. Der Vorteil liegt auf der Hand: Es lassen

sich über eine XML Datei ganze Netzwerke definieren und mit einem Aufruf aufstarten.

1.4.1 Grundlegendes

Auf der VNUML Homepage kann ein bereits konfigurierter Kernel und ein konfiguriertes

Dateisystem heruntergeladen werden. Beides wurde stark optimiert. So ist der Kernel

lediglich 2.2 MB und das Dateisystem 6.6 MB gross.

Folgende Parameter werden zum Starten/Stoppen von Netzwerken benötigt:

Start: ./vnumlparser.pl -t <xml Datei> -v

Stopp: ./vnumlparser.pl -d <xml Datei> -v

Ein Beispiel einer XML Datei mit einem Host:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/share/xml/vnuml/vnuml.dtd"
[
  <!ENTITY VNUMLDIR "/usr/share/vnuml/">
]>
<vnuml>
  <global>
    <version>1.8</version>
    <simulation name>simple</simulation name>
    <automac/>
    <vm_mgmt type="none"/>
    <vm_defaults>
      <filesystem type="cow">
        &VNUMLDIR;filesystems/root_fs_tutorial</filesystem>
      <kernel>&VNUMLDIR;kernels/linux</kernel>
      <console id="1">xterm</console>
```

```

    </vm_defaults>
  </global>
  <!-- NODES -->
  <vm name="host1"/>
</vnuml>

```

Das Attribut im Tag `<filesystem type="cow">` bewirkt, dass das Dateisystem nicht komplett kopiert wird, sondern nur die Änderungen für jede VM abgespeichert werden (in `/etc/vnuml`).

1.4.2 Test A: 1 VM

UML startet viel schneller als die VMs von VMWare. Ebenfalls ist die Ressourcenbelastung wesentlich kleiner:

Eigenschaft	Wert
Grösse des Kernels	2.2 MB
Dateisystem	6.6 MB
RAM	25 MB
Benötigte Zeit zum Aufstarten	6s

1.4.3 Test B: 50 VMs

Leider können in VNUML die Hosts nur sequentiell aufgestartet werden. So wird die CPU nie voll ausgelastet. Für grössere Szenarios mit mehr als 2000 VMs stellt dies ein Problem dar. Es existieren aber Optimierungsmöglichkeiten, welche nach dem Prototyp in Woche 7 evaluiert werden.

Eigenschaft	Wert
Grösse des Kernels	2.2 MB
Dateisystem	6.6 MB
RAM	20 MB pro VM
Benötigte Zeit zum Aufstarten	192s

1.5 QEMU

QEMU ist eine freie virtuelle Maschine. Sie erreicht laut Wikipedia eine gute Ausführungsgeschwindigkeit. Alle wichtigen CPU-Typen werden unterstützt. Wie VMWare läuft es auf allen gängigen Betriebssystemen.



Unter Linux, BSD und Mac OS X unterstützt QEMU auch die Userspace-Emulation. Diese API-Emulation ermöglicht es, dass ausführbare Programme, die für andere dynamische Bibliotheken kompiliert wurden, im Userspace betrieben werden können.

1.5.1 Grundlegendes

Der Benutzer aurel32 von der Debian Community stellt verschiedene Debian Images bereit. In unseren Tests wurde das minimal Image verwendet. Klonen von Images ist problemlos möglich. QEMU lässt sich komplett über eine Konsole ansteuern.

1.5.2 Test A: 1 VM

QEMU startet im Vergleich zu den anderen Alternativen sehr träge auf:

Eigenschaft	Wert
Grösse des Kernels	1.5 MB
Dateisystem	571 MB
RAM	64 MB
Benötigte Zeit zum Aufstarten	50s

1.5.3 Test B: 50 VMs

Beim Versuch, 50 VMs aufzustarten, reagierte das Wirtssystem nach 30 Minuten nicht mehr. Deswegen beziehen sich die Testergebnisse auf ein geändertes Szenario mit 10 VMs parallel:

Grösse des Kernels	1.5 MB
Dateisystem	571 MB
RAM	64 MB pro VM
Benötigte Zeit zum Aufstarten	480s

1.6 Xen

Xen entstand an der Universität Cambridge und wird von XenSource weiterentwickelt. Xen ist ein Hypervisor Typ 1, das heisst es läuft direkt auf der Hardware ohne Hostbetriebssystem. Um die virtualisierten VMs wirklich voneinander zu trennen, benötigt Xen Hardwareunterstützung, beispielsweise Intel VT oder AMD-V. Mit dieser Hardware müssen die Betriebssysteme nicht angepasst werden, und sie merken nicht, dass sie die Hardware teilen.

Die Entwickler haben mit XenSource ein Unternehmen gegründet, das Xen zum Industriestandard machen soll. XenSource wurde im August 2007 für 500 Millionen Dollar durch die Firma Citrix übernommen.



1.6.1 Grundlegendes

Das Dateisystem, in welchem der Kernel direkt integriert ist, kann über eine Konfig-Datei definiert werden. Das Programm `xen-create-image` liest sich die Daten aus und erstellt das Image mit Hilfe von `debootstrap`. Für jede VM, muss dieser Vorgang einmal gemacht werden. Klonen ist nicht möglich. Somit ist das Erstellen eines Pools mit vielen Images initial mit viel Hardware-Aufwand verbunden.

1.6.2 Test A: 1 VM

Die Xen VMs starten extrem schnell:

Eigenschaft	Wert
Grösse des Kernels	1.4 MB
Dateisystem	400 MB
RAM	30 MB
Benötigte Zeit zum Aufstarten	3s

1.6.3 Test B: 50 VMs

Auch Parallel starten die Xen VMs sehr schnell auf. Die Belastung/Aufstartzeit verhält sich ziemlich linear:

Eigenschaft	Wert
Grösse des Kernels	1.4 MB
Dateisystem	400 MB pro VM
RAM	30 MB pro VM
Benötigte Zeit zum Aufstarten	143s

1.7 Kernel-based Virtual Machine

Kernel-based Virtual Machine, kurz KVM wurde im Oktober 2006 veröffentlicht und ist ab Version 2.6.20 des Linux-Kernels als Modul enthalten. Entwickelt wurde es von dem israelischen Unternehmen Qumranet, welches im September 2008 von Red Hat aufgekauft wurde.



KVM selbst nimmt keine Emulation vor, sondern stellt nur die Infrastruktur dazu bereit. Ein modifiziertes QEMU ist zur Zeit die einzige Möglichkeit, diese zu nutzen. Nach dem Laden des Modules arbeitet der Linux Kernel selbst als Hypervisor für virtuelle Maschinen.

1.7.1 Grundlegendes

Mit KVM können von einem bereits bestehenden Image in sehr kurzer Zeit Overlay Images erstellt werden. Auf das Base-Image wird nur lesend zugegriffen und im Overlay Image die Differenzen gespeichert. Änderungen im Overlay-Image können bei Bedarf in das Base-Image comitted werden.

Als Parameter können MAC Adressen für die virtuelle Netzwerkkarte definiert werden. Desweiteren kann eine Multicast Adresse für die VM per Parameter definiert werden.

1.7.2 Test A: 1 VM

KVM VMs benötigen immer mindestens 51 MB RAM, auch wenn weniger für die Instanz konfiguriert wurde.

Eigenschaft	Wert
Grösse des Kernels	1.5 MB

Dateisystem	600 MB
RAM	51 MB
Benötigte Zeit zum Aufstarten	12s

1.7.3 Test B: 50 VMs

KVM benötigt mehr Zeit zum Aufstarten als die anderen Varianten. Trotzdem befindet sich die benötigte Zeit in einem guten Rahmen.

Eigenschaft	Wert
Grösse des Kernels	1.5 MB
Dateisystem	600 MB + 5 MB pro VM
RAM	51 MB pro VM
Benötigte Zeit zum Aufstarten	502s

1.8 Performance Test

Neben den benötigten Ressourcen und der Verwaltbarkeit zählt die Ausführungs­geschwindigkeit ebenfalls zu den wichtigen Punkten der Anforderung: Je schneller sie ist, desto besser skaliert die Lösung.

Die Ausführungs­geschwindigkeit ist grundlegend von zwei Faktoren abhängig: Die Geschwindigkeit der CPU und dem Datendurchsatz im RAM. Um diese beiden Faktoren vergleichen zu können, wurde ein Stressprogramm entwickelt, welches im Anhang zu finden ist. Es berechnet parallel mit 2 Threads die Wurzeln von 1 bis 100'000'000, anschliessend werden 10 mal hintereinander 5/20 MB im RAM linear kopiert. Die Zeit der Threads für die Berechnung und die Durchschnittsgeschwindigkeit der Kopiervorgänge werden auf der Konsole ausgegeben.

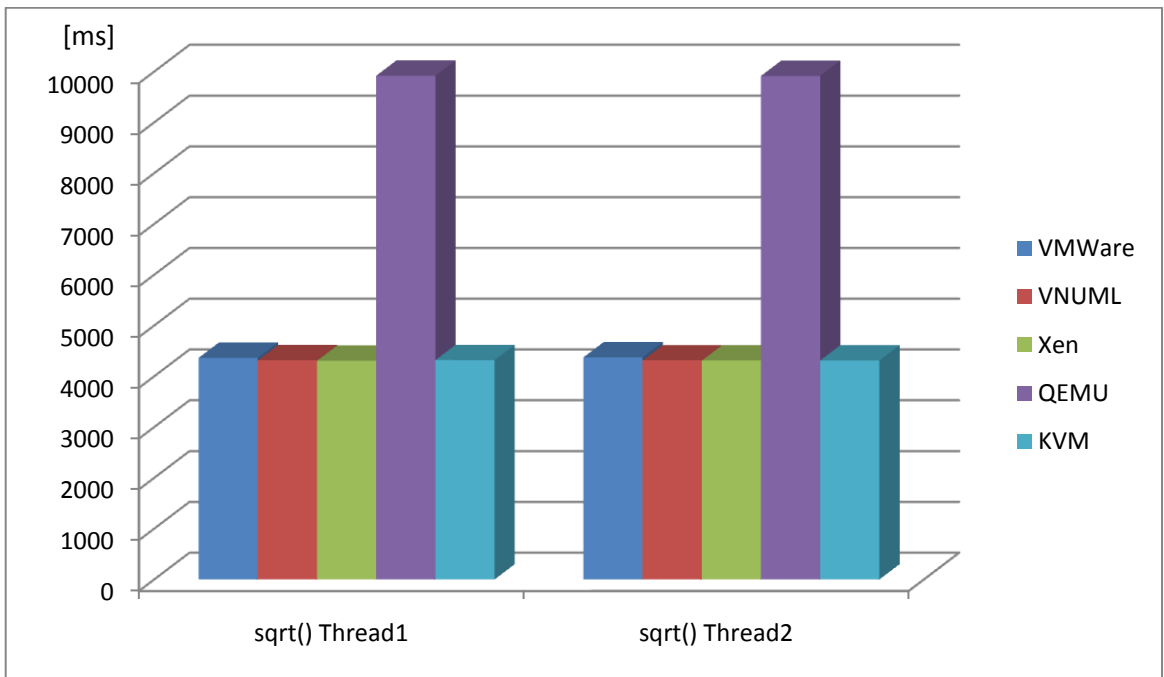


Abbildung 12: Zeitmessung `sqrt()`

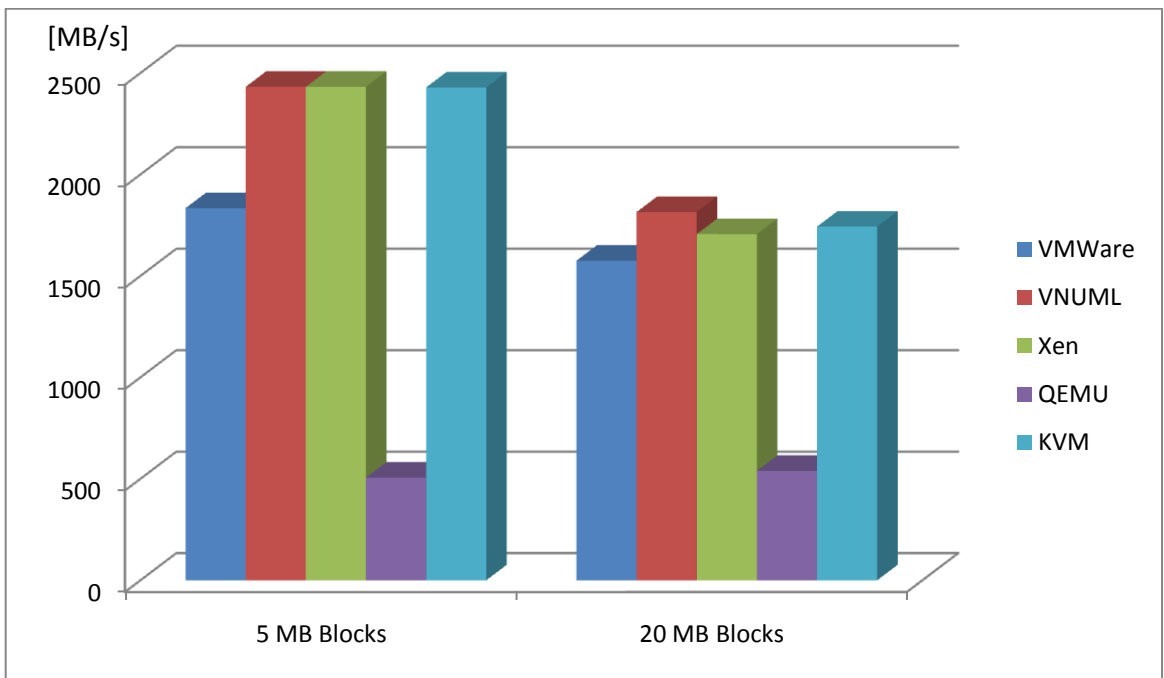


Abbildung 13: Memory Test

1.9 Vergleich

	VMWare	UML	Xen	QEMU	KVM
Kernelgrösse	0.92 MB	2.2 MB	1.4 MB	1.5 MB	1.5 MB
Grösse des Dateisystems	ca. 600 MB	6.6 MB	Ca. 400 MB	571 MB	Ca. 400 MB
RAM Belastung pro VM nach Bootvorgang	54 MB	36 MB	44 MB	64 MB	51 MB
Klonen	Möglich: Option „linked“	Möglich: Attribut type auf „cow“ setzen	-	Möglich	Möglich
CLI/API zur Verwaltung	CLI vorhanden (<i>vmrun</i>); API wäre ebenfalls vorhanden	Kein CLI, XML Dateien müssen Parser übergeben werden	CLI, Konfiguration in XML Dateien	Alles über CLI gesteuert	Komplett über CLI gesteuert
Kommunikation mit Wirtsystem	Virtuelles Netzwerk	Virtuelles Netzwerk	Virtuelles Netzwerk	Virtuelles Netzwerk	Virtuelles Netzwerk
Benötigte Zeit zum Starten von 1 VM	35s	6s	3s	50s	12s
Benötigte Zeit zum Starten von 50 VMs	Nach 12 VMs blockiert VMWare	192s	143s	Hostsystem reagiert nach 30min nicht mehr	502s
100 Mio. Sqrt() Thread1/Thread2	4363ms / 4316ms	4316ms / 4315ms	4308ms / 4311ms	9919ms / 9916ms	4319ms / 4313ms
RAM Durchsatz 5/20 MB Blocks	1834 MB/s 1575 MB/s	2392.00 MB/s 1615.27 MB/s	2432.00 MB/s 1706.67 MB/s	505.82 MB/s 538.95 MB/s	2429.0 MB/s 1743.9 MB/s
Pros	keine	Kompaktes Dateisystem Sehr gute Verwaltbarkeit mit XML Dateien Schnelles Aufstarten	Extrem schnelles Aufstarten Sehr gute Verwaltbarkeit mit Konfig-Dateien Gute Performance	Leicht bedienbar Gute Dokumentation der CLI	Leicht bedienbar Open Source Klonfunktion
Cons	Hohe RAM-Belastung Verwaltung von Teams/Storages für viele VMs fast unmöglich Nach 10 laufenden VMs blockiert VMWare	VMs in XML Datei starten sequentiell	Klonen von VMs nicht möglich	Startet langsam auf Hohe RAM-Belastung Performance schlecht	Benötigt min. 51 MB RAM

Der Performancetest fällt sehr gleichmässig aus, lediglich QEMU zeigte deutlich seine Schwäche.

VMWare ist durch den hohen Verwaltungsaufwand keine gute Wahl. Des Weiteren hat VMWare Mühe, wenn ein Image mehr als 100 mal geklont wird.

VNУML hat durch die Performance und vor allem durch die sehr gute Verwaltbarkeit überzogen. Leider können VMs nur sequentiell vom Parser gestartet werden, was bei mehreren VMs doch einige Zeit in Anspruch nehmen kann. Übrig bleiben Xen und KVM. Xen benötigt für das Basissystem 7 MB weniger RAM als KVM. Allerdings fehlt die Klonfunktion ganz. 7 MB RAM pro VM mehr sind ein guter Kompromiss, welcher für die zusätzliche Klonfunktion von KVM eingegangen werden kann.

KVM schloss sehr gut im Performancetest ab. Sämtliche Einstellungen lassen sich über die CLI einstellen und sind gut dokumentiert. Des Weiteren überzeugen die Overlay-Images sehr. Sie sind einfach verwaltbar, sparen viel Speicherplatz ein und sind sehr schnell erstellt. Aus diesen Gründen wird KVM für die Virtualisation eingesetzt.



2 Programmiersprache / Framework

Eine verteilte Applikation zu programmieren stellt verschiedene Anforderungen an die Programmiersprache. Im Studium wurde hauptsächlich mit den beiden Programmiersprachen Java und C# programmiert. Als Skriptingsprache kamen Bashscripts zum Einsatz.

Um die Kommunikation zwischen verschiedenen Prozessen, welche sich auf verschiedenen physikalischen Maschinen befinden können, zu realisieren, soll vorzugsweise ein ausgereiftes Framework eingesetzt werden. Solche Frameworks werden grundsätzlich in Hochsprachen umgesetzt, da diese entscheidende Vorteile gegenüber Skriptingsprachen bieten. In Hochsprachen existiert eine viel flexibleres Prozess- und Threadmanagement. Darüber hinaus sind Locking-Strategien besser umzusetzen als in Skriptingsprachen.

Ein weiterer Vorteil ist das Error-Left principle von Hochsprachen. Code wird bereits während dem Kompilieren auf Fehler überprüft und nicht erst zur Laufzeit.

Entwicklungswerkzeuge wie Eclipse oder Visual Studio sind ein weiterer wichtiger Faktor, denn sie können den Entwicklungsprozess durch Hilftools massgeblich beschleunigen. Der Entwickler muss sich so nur noch um die „wesentlichen“ Sachen kümmern. Java und C# Programmierer haben beide eine sehr gute Entwicklungsumgebung zur Verfügung, welche verglichen mit den Tools um Shellscripsts zu programmieren, weit überlegen sind.

Durch die benötigte Interprozesskommunikation über die Computergrenze hinweg macht es wenig Sinn, Shellscripsts für die Kommunikation einzusetzen. Trotzdem wäre es denkbar, dass die Remotingschnittstelle Methoden zur Verfügung stellt, um Shellscripsts für kleinere Aufgaben auszuführen. Diese können problemlos durch das verwendete Framework gewrapped werden.

Java und .NET bieten ein ausgereiftes Framework für Remote Procedure Calls (RPC). In Java wird das Framework Remote Method Invocation (RMI) genannt. Das Gegenstück im .NET Framework heisst .NET Remoting und ist im System.Runtime.Remoting Namespace zu finden. Java RMI bringt einige Komplikationen mit dem ClassLoader / Global- & Local-Registry, welche bei .NET Remoting nicht vorhanden sind, da keine Registry zur Speicherung der Netzwerkdienste benötigt wird. Der C# Compiler ist dem Java Compiler bis dato überlegen, was sich in weniger Runtime Ressourcen widerspiegelt und zu geringerer CPU Last führt.

Aus den genannten Gründen sind wir zum Entschluss gekommen, das .NET Framework einzusetzen. Das Mono-Projekt macht es möglich, die komplette .NET 2.0 Runtime plattformunabhängig zu verwenden. So laufen .NET Applikationen auch auf Linux, Solaris und MAC Computer.



3 Analyse Surfers

In der Realität existieren unterschiedliche Benutzer, welche das Portal benutzen. Die Idee ist es nun, verschiedene Typen von Clients zu definieren, welche ein unterschiedliches Surfverhalten vorweisen. In einer ersten Analyse stellen wir eine Liste mit möglichen Eigenschaften zusammen. Ob alle wirklich umsetzbar sind, wird sich in einer späteren Phase bei der Implementierung zeigen. Das System wird so aufgebaut, dass auch nach dem Ende des Projekts, neue Clientcharaktere erstellt und hinzugefügt werden können (DLLs). In einem Szenario wird definiert, wie viele und von welchem Typ vorkommen.

3.1 News-Leser

Der News-Leser loggt sich beim MPP ein. Danach liest er einen Newsartikel von einer Newsseite und meldet sich gleich wieder vom MPP ab. Dadurch wird durch die vielen Sessions, welche auf- und abgebaut werden, das MPP belastet.

3.2 Googler

Ein Googler ist jemand, der nach Begriffen sucht und relativ schnell von Seite zu Seite wechselt, bis er seine gesuchte Information gefunden hat. Dabei kann er mehrere Google Resultate schnell absuchen (und bei denen ein wenig in die Tiefe hineingehen – wenn nicht fündig nächstes Google Resultat absuchen). Der Aufenthalt auf einer Seite ist wesentlich kleiner als beim News-Leser (ca. 10 Sekunden pro Seite bis zum Ziel). Der Unterschied zum News-Leser besteht darin, dass er sich nicht so oft beim MPP an- und abmeldet.

3.3 Downloader

Der Downloader lädt eine Datei von einer URL herunter. Diese kann vor dem Starten des Szenarios festgelegt werden.

3.4 MPPHammer

Der MPPHammer meldet sich beim MPP nicht an. Er versucht trotzdem, verschiedene Seiten aufzurufen. Das MPP leitet ihn jedesmal auf die Landing Page um.

3.5 WebSiteTrasher

Der WebSiteTrasher macht mehrere Zugriffe asynchron auf eine Webseite. Dadurch kann neben dem MPP auch ein Webserver getestet werden, wie er mit vielen simultanen Anfragen umgehen kann.

4 Prototyping Control Center

4.1 1st Draft

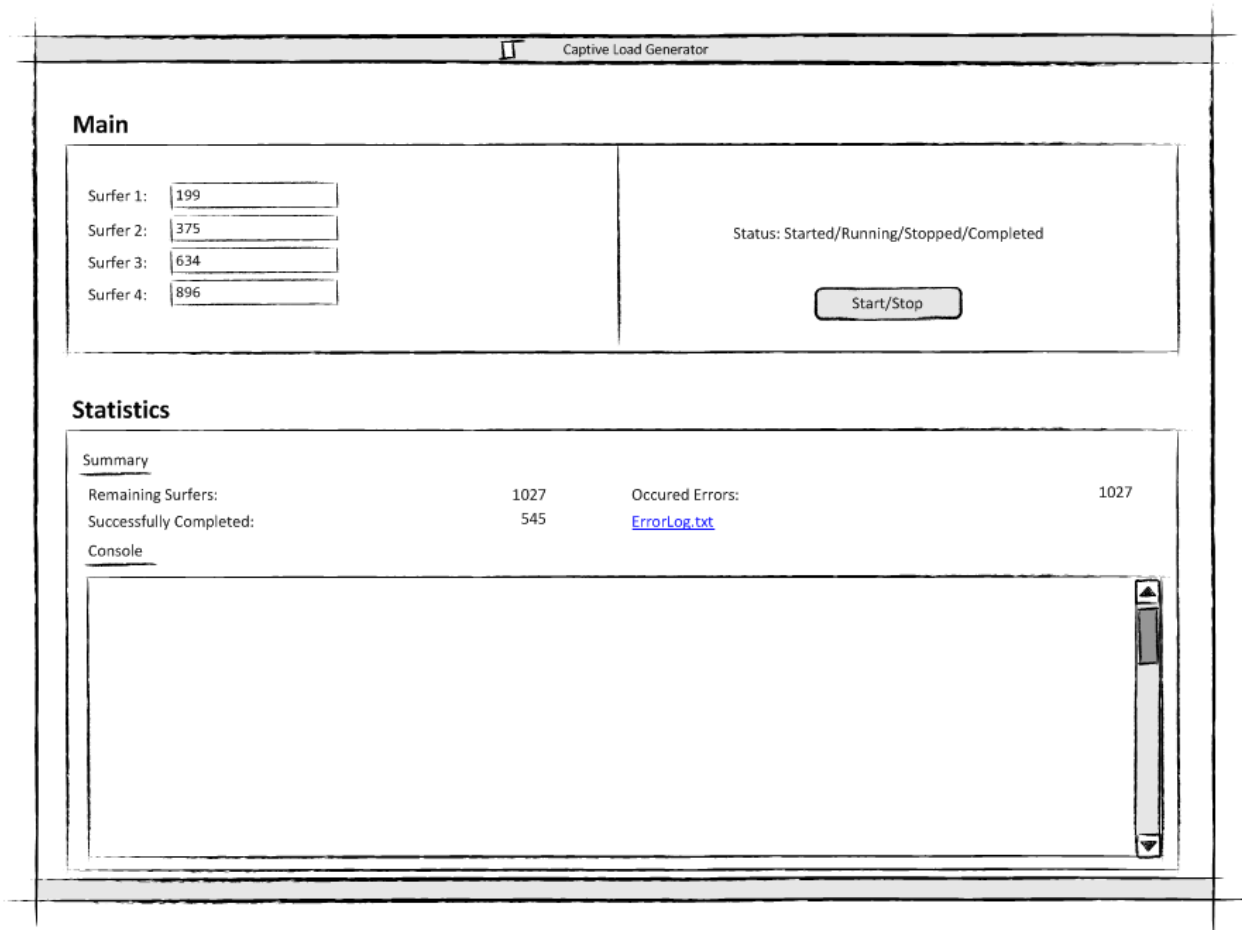


Abbildung 14: 1. Visio Skizze von dem Control-Center GUI

4.2 2nd Draft

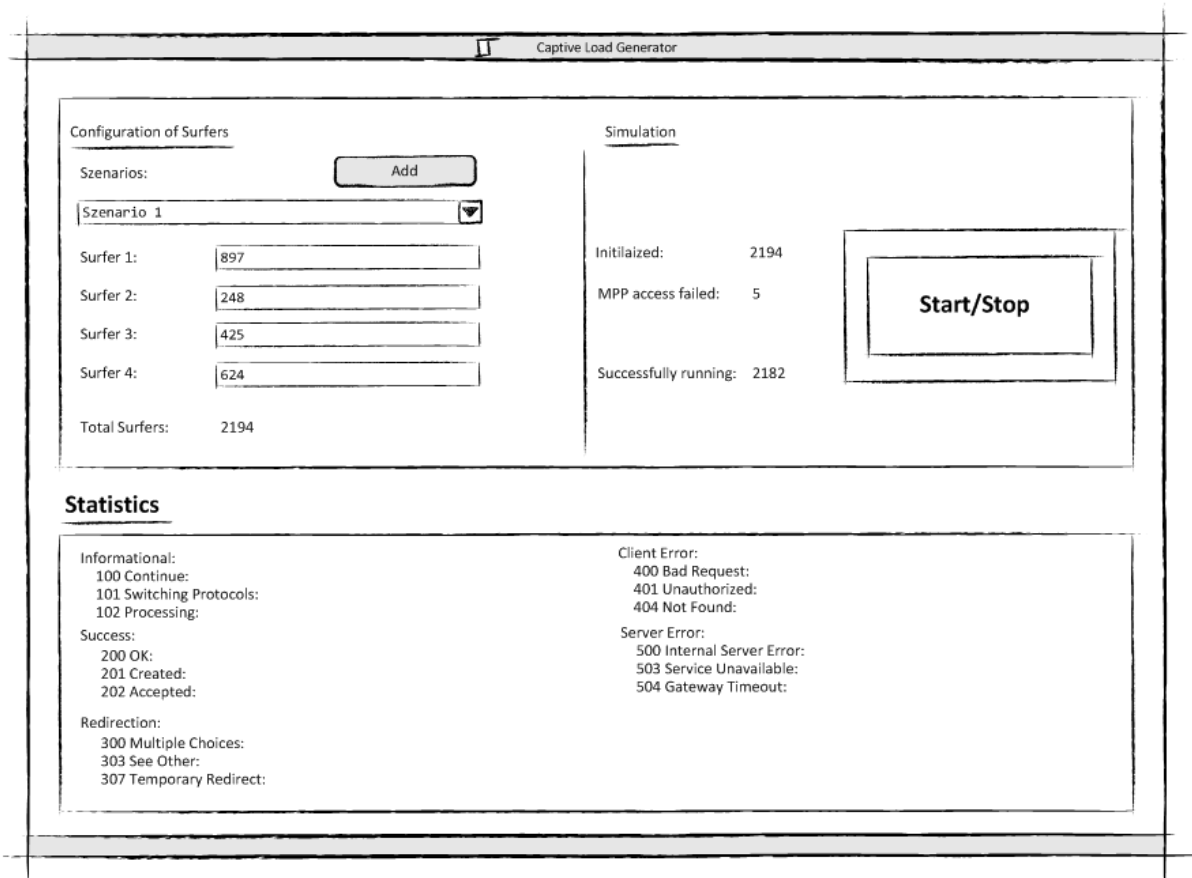


Abbildung 15: 2. Visio Skizze von dem Control-Center GUI

Nach der ersten Besprechung wurden Verbesserungen und Wünsche vorgeschlagen, welche in den zweiten Entwurf übernommen wurden.

Die Verbesserungen und Änderungen zum ersten Entwurf:

- Es braucht keinen Konsolenoutput
- Detailliertere Statistiken
- Möglichkeit verschiedene Szenarien zu speichern und konfigurieren

4.3 3rd Draft

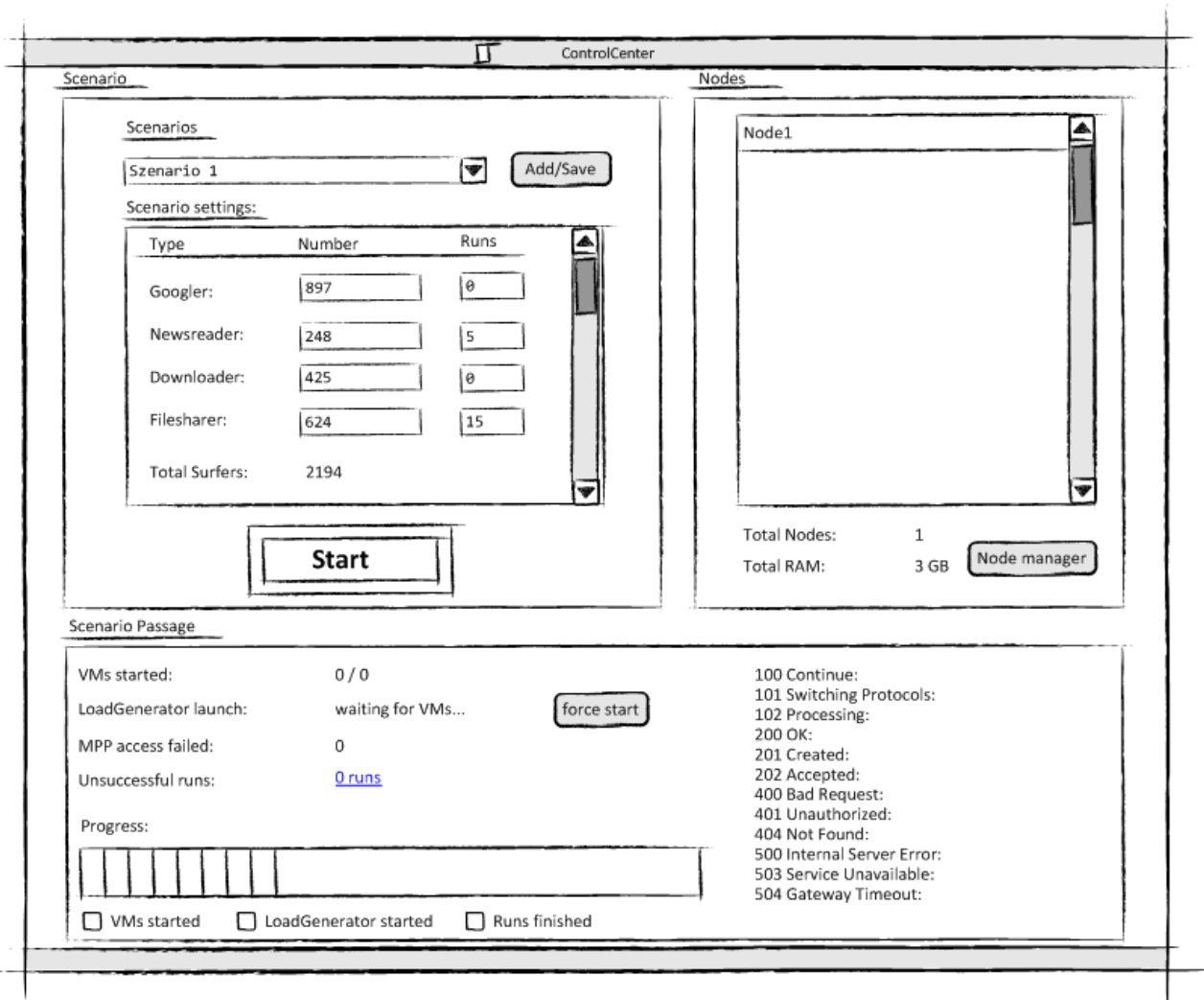


Abbildung 16: 3. Visio Skizze von dem Control-Center GUI

Änderungen zum zweiten Entwurf:

- Durch eine Anforderungsänderung kam eine Zusatzübersicht, in welcher die verbundenen Knoten zu sehen sind
- Szenario soll auch gelöscht werden können
- Progressbar um einen schnellen Überblick über den aktuellen Fortschritt des Szenarios zu haben

4.4 Finaler Entwurf

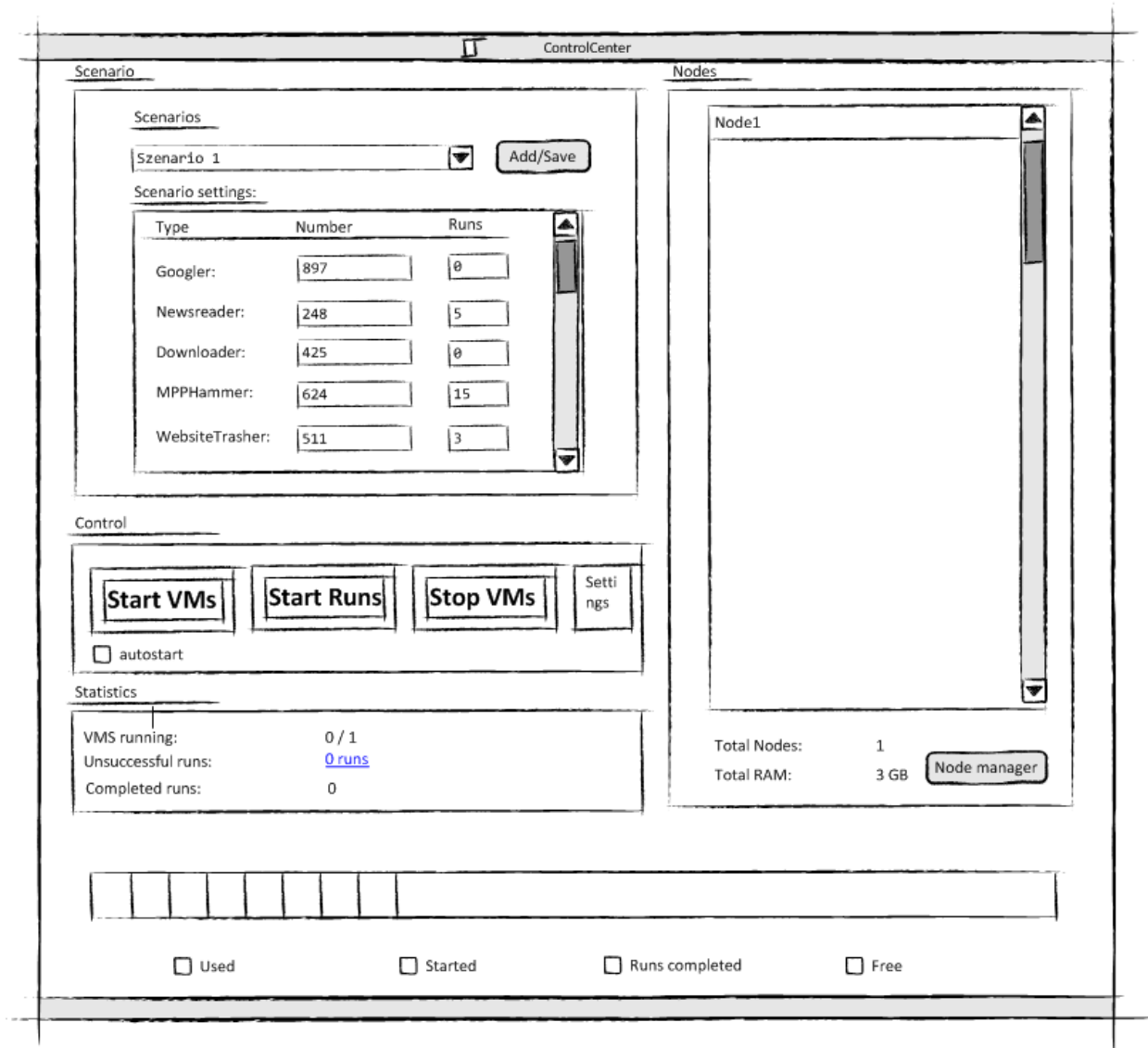


Abbildung 17: Finale Skizze von dem Control-Center GUI

Änderungen zum dritten Entwurf:

- HTTP Statuscodes wurden ganz rausgenommen, weil diese in ein Logfile geschrieben werden. Dies führt zu einer besseren Übersicht und überfüllt die Sicht nicht.
- Die Nodes Übersicht wurde vergrößert, da im Praxistest aufgefallen ist, dass die Übersicht wertvoll ist.
- Zusätzliche GUI-Elemente sind dazu gekommen, um die Funktionen, wie z.B. Settings, zu unterstützen

5 MPP

5.1 Funktionsweise

Das MPP identifiziert die Clients auf Layer 2. Wenn sie noch nicht eingeloggt sind, werden sie beim ersten Webseitenaufruf auf eine Landingpage umgeleitet. Auf dieser Seite muss sich der Client authentifizieren und gegebenenfalls die AGBs akzeptieren:

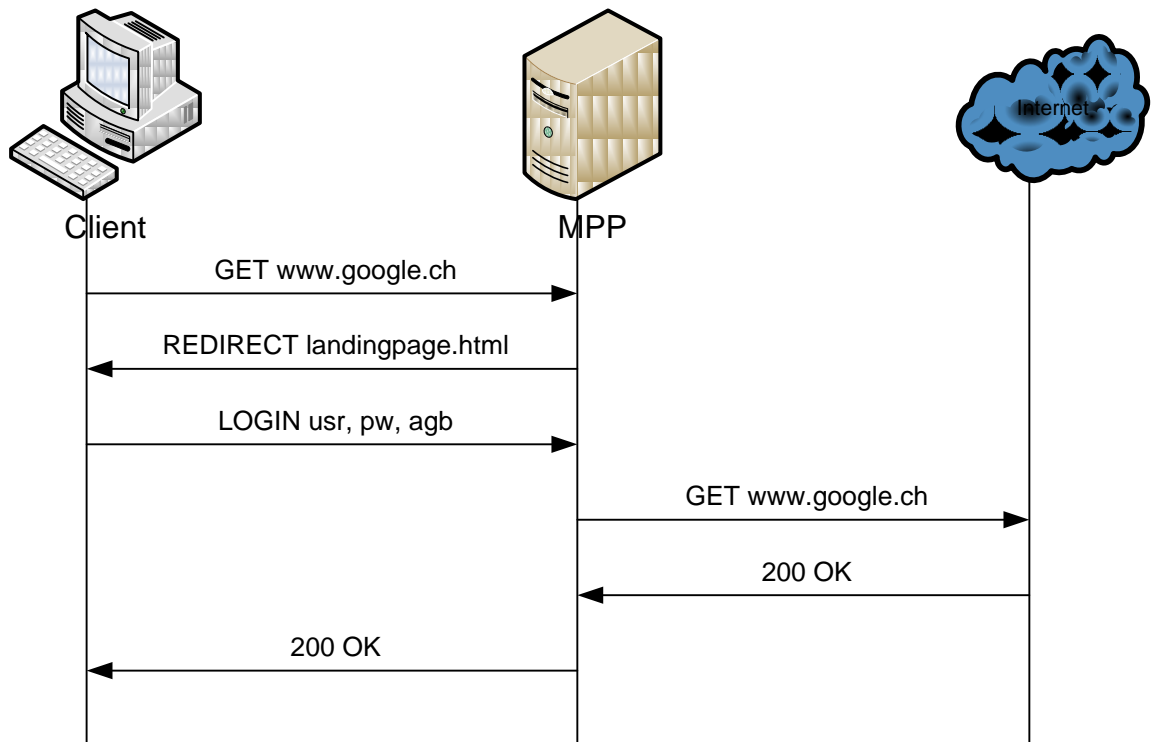


Abbildung 18: Beispiel einer Kommunikation eines Clients mit dem MPP

6 Design & Architektur

6.1 Architektur Übersicht

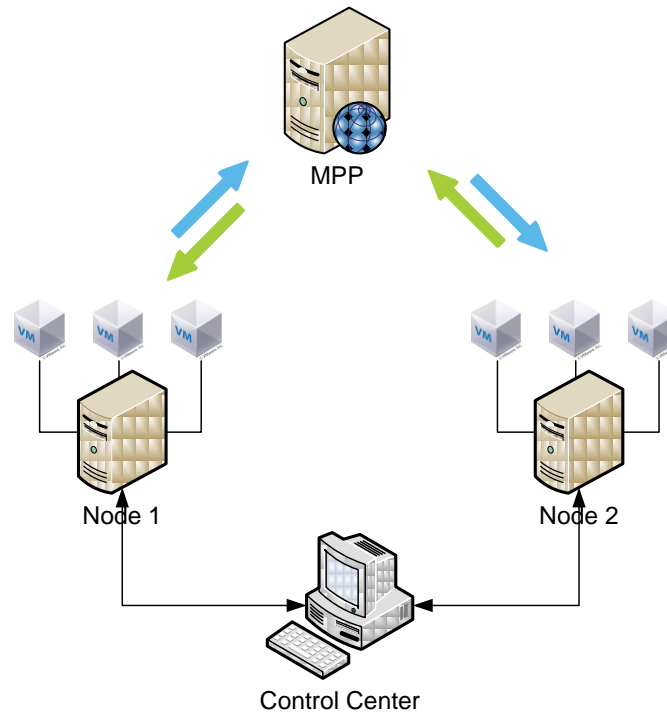


Abbildung 19: Architekturübersicht

Es handelt sich um ein verteiltes System, welches sich von einem Punkt aus steuern lässt. Bei Bedarf können neue Computer dem System hinzugefügt werden. Dadurch kann die Anforderung Skalierbarkeit erfüllt werden. Das System besteht aus verschiedenen Komponenten, welche wiederum verschiedene Services anbieten. Folgende Tabelle soll einen ersten Überblick geben:

Komponente	Beschreibung
ControlCenter.exe	Das Control-Center kann auf entfernen Computer (sogenannte Nodes) VMs steuern und verwalten. Des Weiteren sammelt es Informationen über den Fortschritt eines Szenarios und zeigt diese dem Benutzer an. Es ist nur einmal im System vorhanden und dient zur Steuerung des gesamten Systems.
VMRemoteControl.exe	Die VMRemoteControl Anwendung befindet sich auf den Nodes und stellt dem Control-Center über das Netzwerk Funktionen bereit, damit es VMs auf dem Computer steuern und verwalten kann.
LoadGenerator.exe	Die LoadGenerator Anwendung ist auf den VMs

	vorinstalliert und wird nach dem Bootvorgang automatisch gestartet. Es simuliert verschiedene Surfertypen. Diese werden dynamisch zur Laufzeit nach der Typenzuweisung via Reflection geladen.
Surfers (.dll)	Die Surfers sind in Form von DLLs auf den VMs vorinstalliert. Die LoadGenerator Anwendung erzeugt zur Laufzeit dynamisch eine Instanz davon. Im Control-Center kann definiert werden, wie viele Durchläufe gemacht werden.

Damit die Komponenten untereinander kommunizieren können, müssen sie sich Services zur Verfügung stellen. Folgende Tabelle soll einen kurzen Überblick geben, welcher Service auf welcher Komponente läuft. Die genaue Schnittstelle der erwähnten Services werden in den folgenden Kapiteln behandelt.

Service	Beschreibung
IRegister (Control-Center)	Durch diesen Service können sich die beiden Applikationen VMRemoteControl und LoadGenerator beim Control-Center bekannt machen. Erst durch die Registrierung weiss das Control-Center, wie viele Computer im Netzwerk am ganzen System beteiligt sind.
IVMControl (VMRemoteControl)	Durch diesen Service kann das Control-Center auf einem entfernten Computer virtuelle Maschinen steuern und verwalten.
IStatus (Control-Center)	Die LoadGenerator Instanzen nutzen den Status-Service des Control-Centers, um Informationen über den Fortschritt mitzuteilen.
ILoadGeneratorControl (LoadGenerator)	Das Control-Center verwendet diesen Service, um den Surfer zu steuern (Start/Stop).

6.2 Deployment

Auf folgendem Diagramm ist das ganze System mit einem Computer als Host für VMs dargestellt.

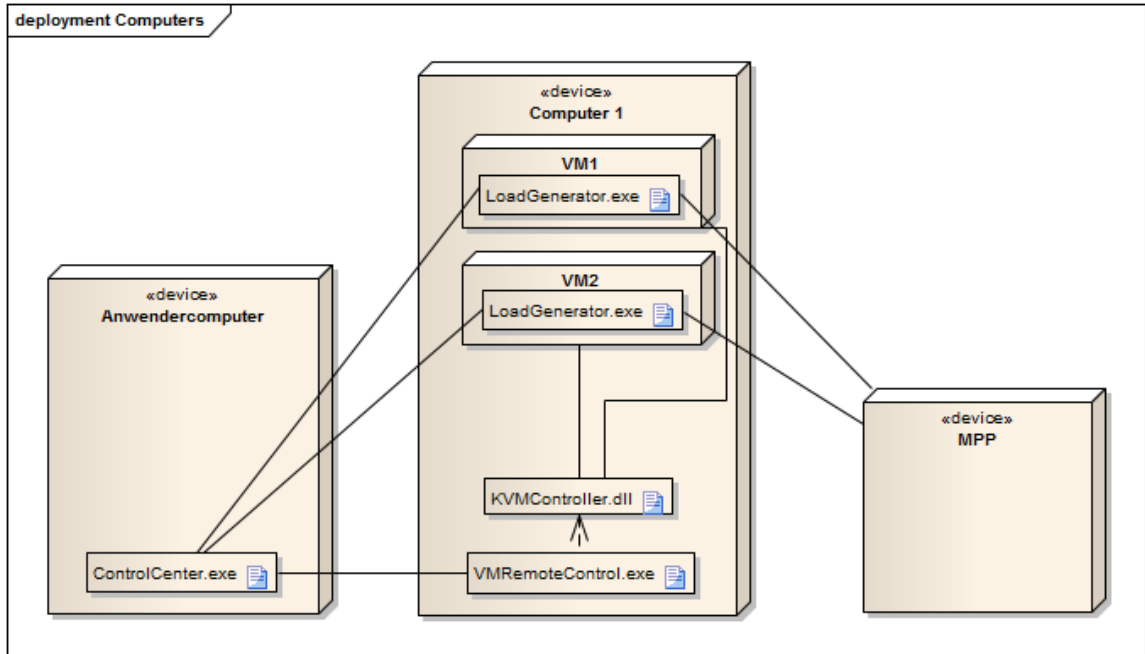


Abbildung 20: Deployment Diagramm

6.3 Services

6.3.1 IRegister - Registrationservice

Der Registrierservice, welcher das Control-Center anbietet, besteht aus 2 Methoden:

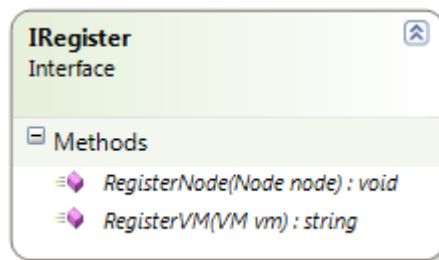


Abbildung 21: Register Service Remote Interface

Mit Nodes sind die Computer gemeint, auf welchen VMs gestartet werden für die Szenarien. Die VMRemoteControl Anwendung (welche auf diesen Computer läuft), registriert sich beim Starten beim Control-Center über diesen Service.

Die VMs werden später von der LoadGenerator Anwendung registriert. Die VMs sind so eingerichtet, dass sie nach dem Bootprozess diese Anwendung automatisch starten. Diese meldet sich dann über den Service beim Control-Center an. Als Rückgabewert erhalten sie einen Surfertypen zugewiesen.

Das Control-Center speichert alle Registrierungen in der Klasse VMManager:

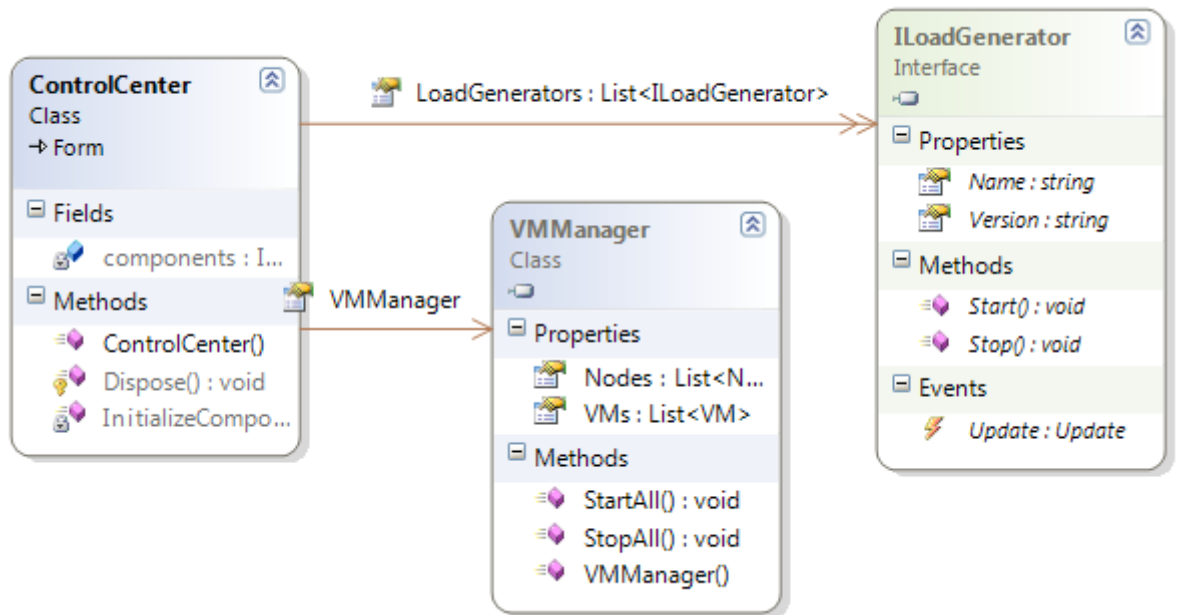


Abbildung 22: Beziehungen Control-Center - VMManager - ILoadGenerator

6.3.2 IStatus - Statistik Reporting Service

Damit die LoadGenerator Anwendung dem Control-Center Statusbenachrichtigungen übermitteln kann, stellt das Control-Center einen Service zur Verfügung, welcher das Interface IStatus implementiert:

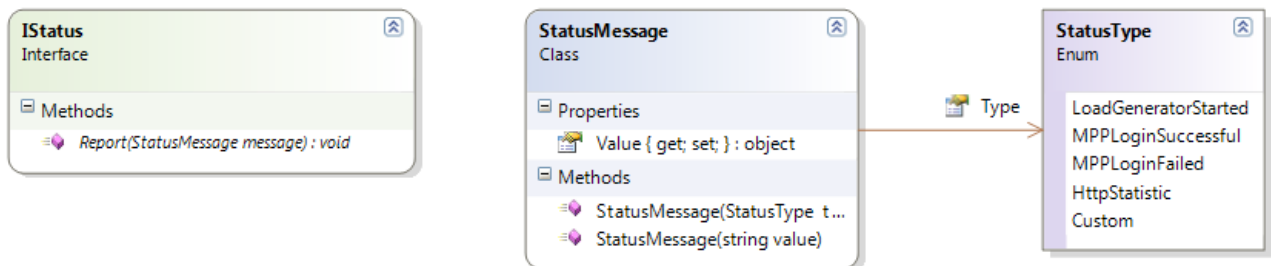


Abbildung 23: Status Remote Interface

6.3.3 IVMControl – VMs steuern und verwalten

Das Control-Center verwendet diesen Service auf den Nodes, um VMs zu steuern und verwalten. Die konkrete Implementation wird als DLL deployed und dynamisch via Reflection beim Starten der VMRemoteControl Anwendung geladen. Dadurch kann die Umsetzung der Virtualisierung ausgetauscht werden, ohne dass andere Komponenten angepasst werden müssen.

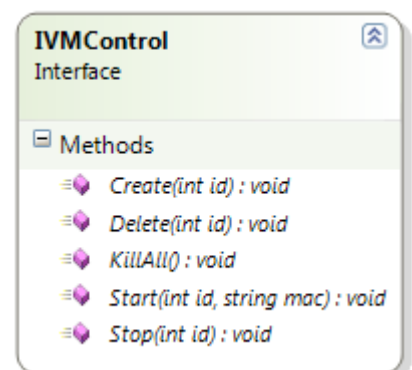


Abbildung 24: Interface zur Steuerung der VMs auf der Node

6.4 Netzwerk / Kommunikation

6.4.1 Grundlegendes

Das MPP identifiziert Clients auf Layer 2. Deswegen ist es notwendig, dass sich zwischen den einzelnen VMs kein Router befindet. Auf den Nodes muss dazu ein Bridge-Interface konfiguriert werden.

Zur Kommunikation wird das .NET Remoting Framework eingesetzt. Das Framework verwendet TcpChannels, um zu kommunizieren. Wie der Name bereits sagt, wird TCP verwendet. UDP wird durch das Framework nicht unterstützt. Der Startbefehl für alle LoadGenerator Instanzen soll aber per UDP Multicast gesendet werden. Dies ist die einzige Kommunikation, welche von „Hand“ implementiert wird, da sich dies mit Multicast um einiges besser umsetzen lässt. Auf diesen Spezialfall wird im übernächsten Kapitel eingegangen

6.4.2 Node Discovery

Die Node Discovery ist per Multicast gelöst. Auf der Adresse **224.2.2.2** und dem Port **1400** ist die VMRemoteControl Applikation erreichbar. Der Service zur Steuerung der VMs (VMRemoteControlService) kann über diese Multicast Adresse gestartet werden. Ebenfalls ist es möglich, den Service komplett neu zu starten (dabei werden alle KVM Prozesse beendet). Beim Start des Services meldet sich dieser beim Control-Center an und übermittelt Informationen wie die Anzahl Cores und der Verfügbare Arbeitsspeicher. Da das Control-Center den Service auf den Nodes startet, steht im Multicast Packet als Absender die IP Adresse des Control-Centers drin. Somit muss auf den Nodes keine Konfiguration zur Findung des Control-Centers vorgenommen werden, denn diese können die IP aus dem Multicast Packet auslesen.

6.4.3 Beispielsequenzdiagramm

Folgendes Diagramm zeigt einen Beispielsablauf von der Registrierung bis das Szenario läuft:

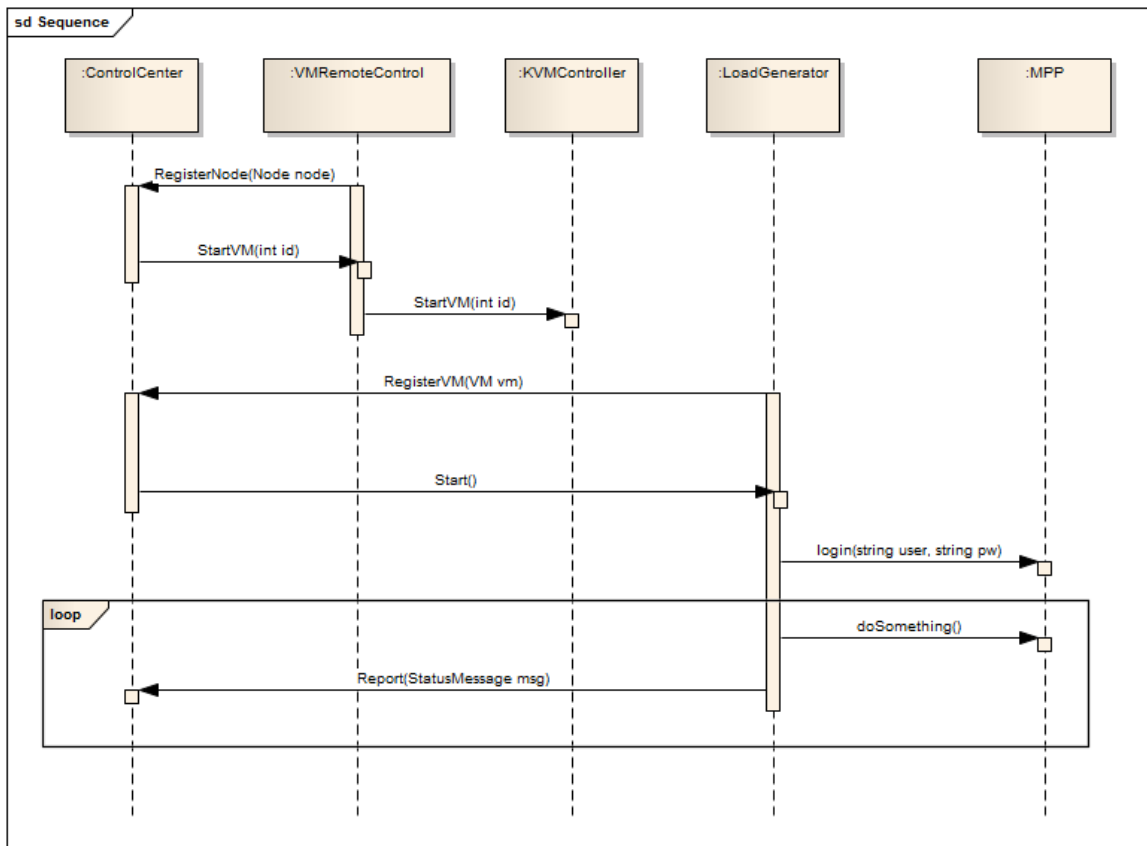


Abbildung 25: Interaktion aller Komponenten

6.4.4 Startbefehl per Multicast

Für den Startbefehl wird UDP verwendet. Dies hat den Vorteil, dass Multicast eingesetzt werden kann. Somit muss der Startbefehl nur einmal gesendet werden.

6.4.5 Kommunikation untereinander

Beim Captive Load Generator sind sehr viele Computer (auch virtuelle) involviert, welche miteinander kommunizieren und sich somit kennen müssen. Der Konfigurationsaufwand soll trotzdem möglichst klein gehalten werden. Um dieses Kriterium zu erfüllen, ist die Architektur so ausgelegt, dass nur eine IP im ganzen System bekannt sein muss. Diese IP ist die vom Control-Center. So muss lediglich die LoadGenerator Applikation auf den VMs mit der richtigen IP als Parameter ausgeführt werden. Bevorzugterweise kann ein domain name verwendet werden:

```
debian:/root# mono LoadGenerator.exe captive.big0.li
```

6.5 Logische Architektur

Für die logische Architektur wird ein geschichtetes System eingesetzt. Der Vorteil dieses Systems ist, dass einzelne Schichten ausgetauscht bzw. ersetzt werden können, ohne die anderen Schichten zu beeinflussen. Zusätzlich wird durch diesen Architekturansatz die Kopplung der einzelnen Pakete stark reduziert.

Des Weiteren wird stark gegen Interfaces programmiert, welche in der Common-Schicht definiert sind. Dadurch können die Komponenten hinter den Interfaces ausgetauscht werden, ohne dass andere Komponenten beeinflusst werden.

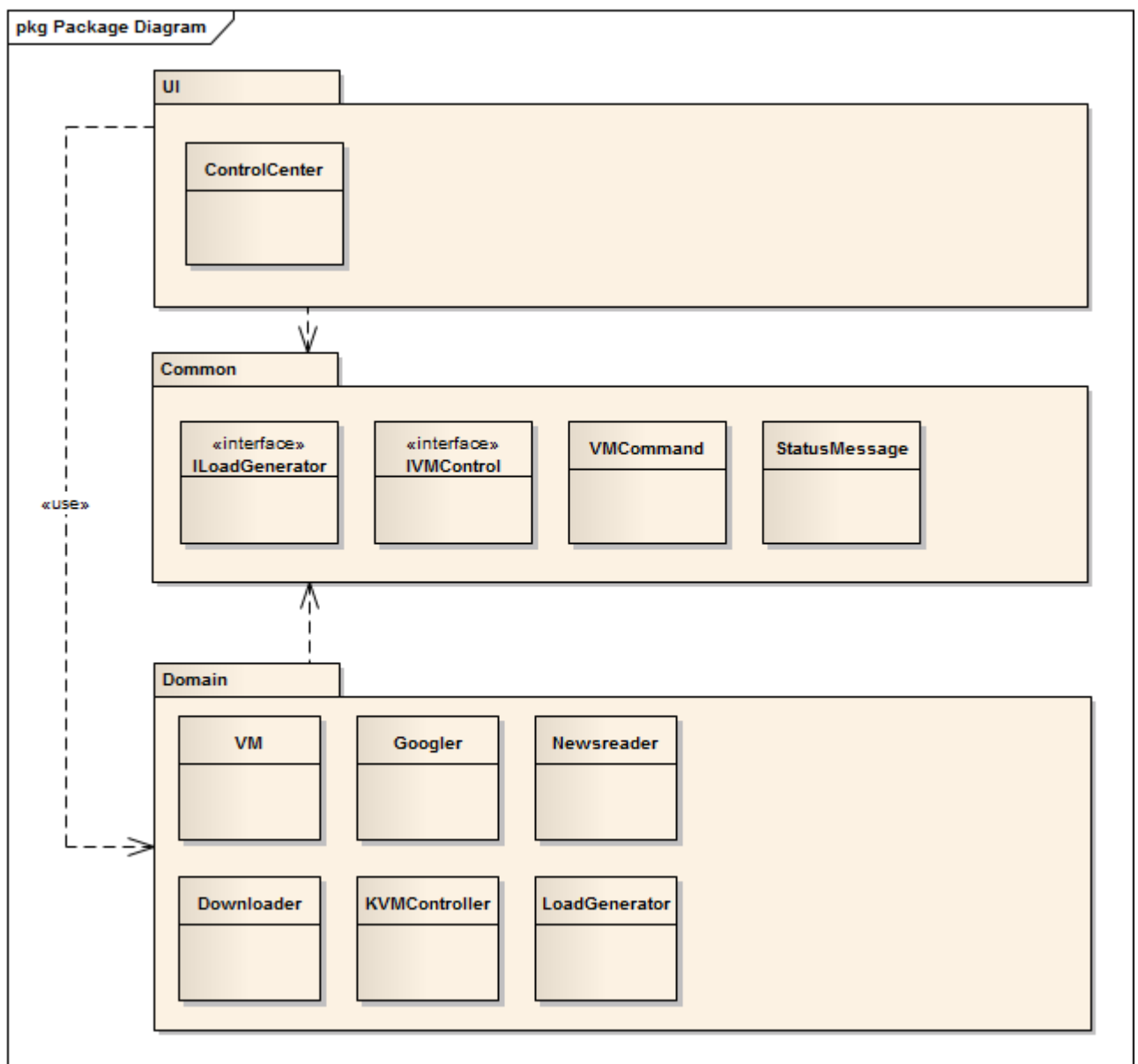


Abbildung 26: Logische Architektur

7 KVM – Kernel based Virtual Machine

Durch die Analyse wurde KVM zur Virtualisation gewählt. In diesem Kapitel wird auf die verwendeten Funktionen von KVM eingegangen. Leider ist die Dokumentation nicht ganz vollständig. Auch sind nicht alle Fragen im FAQ auf der offiziellen Seite beantwortet.

7.1 Images

KVM unterstützt diverse Formate für Images. Neben dem „raw“ Format wird das „qcow2“ Format in diesem Projekt eingesetzt. Es ist ein copy-on-write Format. Wie es der Name sagt, werden Daten kopiert, bevor sie geschrieben werden. Dies macht unter Verwendung eines Base-Images Sinn: aus einem read-only Base-Image werden die Daten beim Ändern in das qcow2 Image kopiert und geändert. So können von einem Base-Image mehrere qcow2-Images erstellt werden. In den qcow2 Dateien werden lediglich die Differenzen gespeichert. Solche qcow2-Images werden auch Overlay-Images genannt. Da das Base-Image read-only gemounted wird, gibt es keine Concurrency Probleme, wenn mehrere Overlay-Images vom selben Base-Image gestartet werden.

```
kvm-img create -f qcow2 base.img 8G
```

Erstellt ein Image, welches bis zu 8GB gross werden kann. Die volle Grösse wird nicht zu Beginn alloziert, sondern wächst automatisch.

```
kvm-img create -b base.img -f qcow2 overlay.img
```

Im overlay.img Image werden nun nur die Differenzen zum base.img Image gespeichert. Wenn Änderungen vom Overlay in das Base Image übernommen werden sollen, kann dies über folgenden Befehl getan werden:

```
kvm-img commit -f qcow2 overlay.img
```

KVM bietet noch diverse weitere Möglichkeiten, welche durch dieses Projekt aber nicht zum Einsatz kommen und deswegen nicht weiter erläutert werden.

7.2 Virtuelle Maschinen starten

KVM fiel bereits zu Beginn durch seine Einfachheit auf. Das erste Mal eine VM zu starten, um die Grundinstallation auszuführen, kann bereits durch sehr wenige Parameter getan werden:

```
kvm -hda image.img -cdrom DebianInstaller.iso -boot d
```

Lediglich das Festplatten-Image der VM und von einer Installations-CD wird benötigt. Der Parameter `-boot d` lässt die VM automatisch von der Installations-CD booten.

Für die einzelnen VMs, welche die LoadGenerators beherbergen, werden ein paar Parameter mehr benötigt:

```
kvm -hda image qcow2 -net nic,macaddr=<MAC> -net tap -m 51
```

Damit wird dem virtuellen Networkinterface eine MAC Adresse zugewiesen. Mit `-m` kann der VM eine bestimmte Menge an RAM zugewiesen werden.

8 Szenarien

Ein Szenario definiert die Surfotypen und wie viele von denen für einen Testlauf benötigt werden. Das Control-Center verwaltet die Szenarien und bietet dem Benutzer eine Maske, um diese bearbeiten zu können.

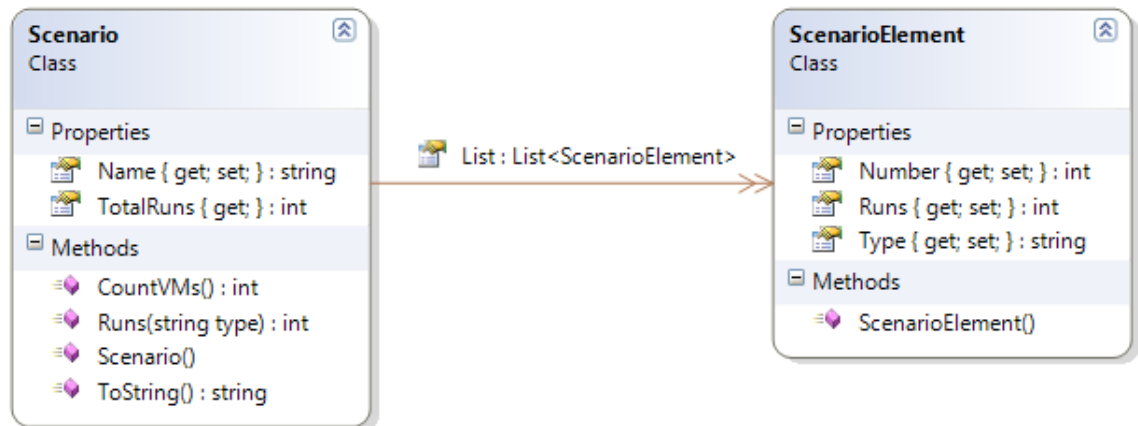


Abbildung 27: Szenario Klassendiagramm

Grundsätzlich hat ein Szenario ein Name und dazu eine Liste von Typen, welche darin vorkommen. Zu jedem Typen kann die Anzahl und wie viele Runs pro Typ gemacht werden, definiert werden.

8.1.1 Serialisierung

Die Szenarien werden beim Benutzer im Eigenen Dateien Ordner in der Datei Scenarios.xml abgelegt. Das XML weist dazu folgende Struktur auf:

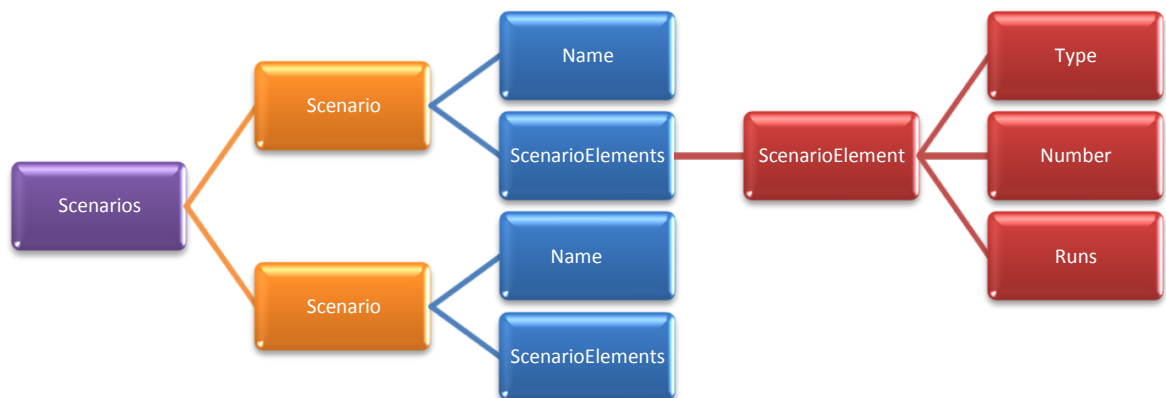
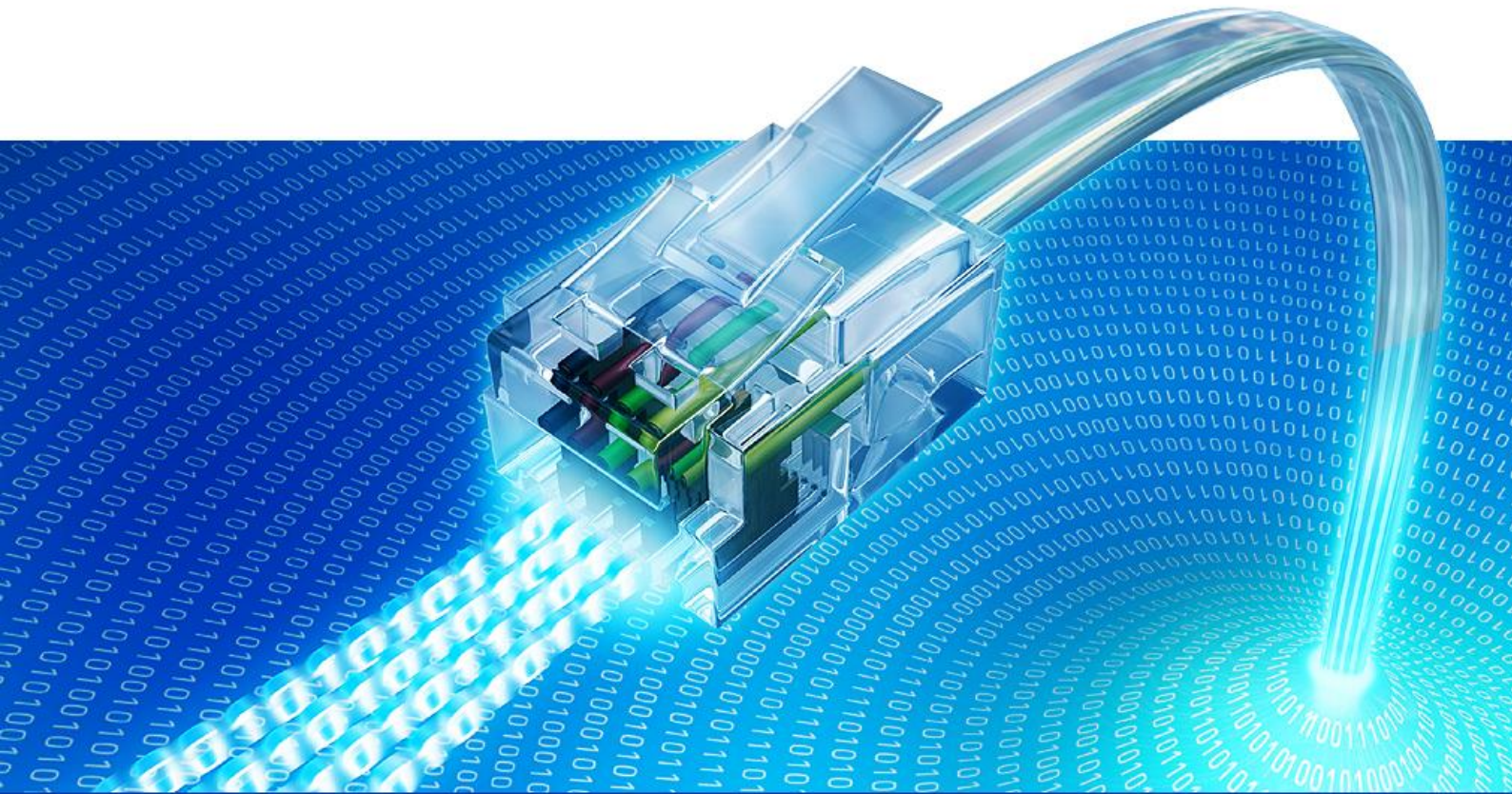


Abbildung 28: XML Struktur der gespeicherten Szenarien



10. Installation

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Installation / Konfiguration

Die Applikation Captive Portal Load Generator besteht aus drei verschiedenen Komponenten, welche individuell installiert und konfiguriert werden müssen. In diesem Kapitel werden diese Komponenten beschrieben. Wo möglich können die Komponente über ein Bash-Script installiert werden.

Komponenten:

- **Control Center**
GUI zur Steuerung der ganzen Applikation. Wird nur einmal benötigt.
- **VMRemoteControl**
Applikation, welche dem Control-Center die lokale Hardware zur Verfügung stellt. Diese Komponente wird in der Form eines vorinstallierten Debian Images ausgeliefert.
- **LoadGenerator**
Der LoadGenerator wird innerhalb mehreren Images gestartet. Diese Komponente wird ebenfalls als vorkonfiguriertes Image ausgeliefert, in welcher die Applikation integriert ist.

1.1 Control-Center

Das Control-Center ist vorzugsweise auf einem Windows XP oder höher Betriebssystem zu installieren. Trotzdem ist es möglich, diese Anwendung auf Linux oder MacOSX zu verwenden. Es kann allerdings zu kleineren Darstellungsproblemen kommen. Falls die Zeit noch reicht während dem Semester, werden diese Probleme noch behoben.

Damit das Control-Center gestartet werden kann, muss auf Windows die .NET 2.0 Runtime vorinstalliert sein. Ab Service Pack 2 (Windows XP) ist die Runtime bereits installiert. Anschliessend kann das Control-Center.zip Archiv auf der Abgabe-CD an ein beliebigen Ort entpackt und die sich darin befindende Control-Center.exe Datei ausgeführt werden.

1.2 VMRemoteControl

Zur Anwendung VMRemoteControl gehört ein vorkonfiguriertes Linux (Debian oder Ubuntu). Die komplette Konfiguration wurde in einem Shellscript geschrieben, damit Änderungen einfach vorgenommen werden können. Das Shellscript ist im Anhang zu finden. Im Rahmen dieser Arbeit wird die Umsetzung der Virtualisierung für KVM implementiert (KVMController.dll). Deswegen kann die Applikation nur auf einem Linux Rechner mit einem Kernel >= 2.6.20 eingesetzt werden.

1.2.1 Benötigte APT Packages

Folgende Pakete müssen über APT installiert sein:

```
bridge-utils uml-utilities kvm mono-runtime libmono-corlib2.0-cil  
libmono-winforms2.0-cil libmono-system-runtime2.0-cil sudo
```

Zusätzlich zu der Anwendung VMRemoteControl.exe wird die DLL Captive.Common.DLL benötigt. Sie beinhaltet unter anderem die Schnittstellen der verschiedenen Services und dient damit als Basis, damit alle Komponenten die Schnittstellen kennen.

1.2.2 Netzwerkkonfiguration

Damit die VMs auf dem Betriebssystem eine IP vom DHCP bekommen, wird eine Bridge konfiguriert. Für jede VM welche gestartet wird, wird ein tap interface erstellt. Dazu ist das Skript `/home/captive/kvm-ifup` zuständig. Es wird beim Start jeder VM einmal ausgeführt. Beim Beenden wird das `kvm-ifdown` Script ausgeführt, welches das tap device wieder löscht.

1.2.3 Benötigte Assemblies

Folgende Assemblies werden benötigt:

- [Captive.Common.dll](#)
Alle Komponenten benötigen diese DLL. Sie beinhaltet unter anderem die Definition der gemeinsam benötigten Schnittstellen und Datentypen.
- [Captive.Virtualisation.KVMController.dll](#)
Implementation der Schnittstelle `IVMControl` für KVM. Wenn eine andere Lösung zur Virtualisation gewünscht ist, kann diese DLL ersetzt werden. `VMRemoteControl` sucht sich automatisch eine DLL im Startup-Pfad und lädt die erste, welche eine Klasse beinhaltet, die das `IVMControl` Interface implementiert. Die Auswahl der DLL/Klasse wird auf der Konsole ausgegeben.
- [VMRemoteControl.exe](#)
Dies ist die Anwendung, welche gestartet werden muss. Sie bietet den Service `VMRemoteControlService` an, welcher vom Control-Center konsumiert wird. Über Multicast kann der Service zurückgesetzt werden.

1.2.4 Installationscript

Das `install.sh` Script führt alle nötigen Schritte zur Installation aus.

1.3 Base-Image LoadGenerators

Das Base-Image (`base.img`) muss auf jeder Node im Verzeichnis `/home/captive` vorliegen. Das Image ist so konfiguriert, dass wenn die Netzwerkschnittstelle bereit ist, automatisch die LoadGenerator Anwendung gestartet wird (`ifup` Script). Diese meldet sich beim Control-Center an und wartet auf Anweisungen.

Mit folgendem Befehl kann das Base-Image für die Konfiguration gestartet werden:

```
kvm -hda /home/captive/base.img -m 51
```

1.3.1 Benötigte APT Packages

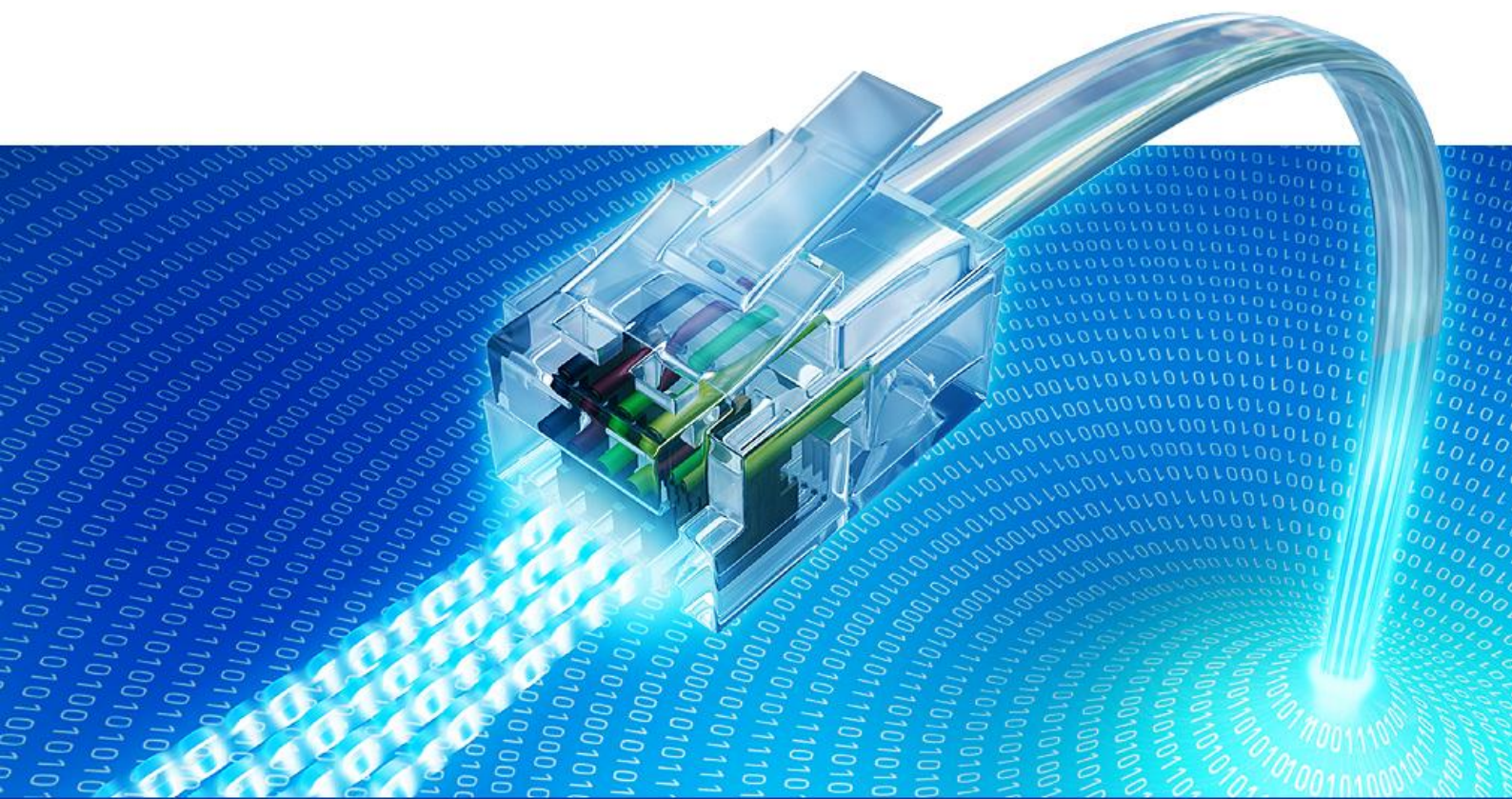
Folgende Pakete müssen über APT installiert sein:

```
bridge-utils uml-utilities kvm mono-runtime libmono-corlib2.0-cil  
libmono-winforms2.0-cil libmono-system-runtime2.0-cil sudo
```

1.3.2 Benötigte Assemblies

Folgende Assemblies müssen im Image vorhanden sein:

- [Captive.Common.dll](#)
Alle Komponenten benötigen diese DLL. Sie beinhaltet unter anderem die Definition der gemeinsam benötigten Schnittstellen und Datentypen.
- [Captive.LoadGenerator.Googler.dll](#)
- [Captive.LoadGenerator.NewsReader.dll](#)
- [Captive.LoadGenerator.Downloader.dll](#)
- [Captive.LoadGenerator.MPPHammer.dll](#)
- [Keywords.txt](#)
Suchwörter für den Googler.
- [Sites.txt](#)
Beinhaltet links zu Newsseiten, welche vom News-Reader verwendet werden



11. API

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Load Generator API

Die Architektur ist so ausgelegt, dass Erweiterungen möglichst einfach implementiert und in das vorhandene System eingebunden werden können. Um neue Surfertypen hinzuzufügen, muss eine neue Klasse erstellt werden, welche das Interface ILoadGenerator implementiert.

Die Klasse muss als DLL im Verzeichnis vom Control-Center und ebenfalls im VM-Image, im selben Ordner wie die LoadGenerator Anwendung vorliegen. Das Control-Center erkennt alle Klassen, welche das Interface implementieren automatisch und zeigt diese im GUI an.

Die LoadGenerator Anwendung bekommt nach der Registrierung der VM beim Control-Center einen Typen zugewiesen. Dieser wird in den DLLs per Reflection gesucht und instantiiert. Somit ist es nicht nötig, die LoadGenerator Anwendung neu zu kompilieren, wenn neue Typen hinzugefügt werden.

1.1 Schnittstelle

Die Schnittstelle besteht aus vier read-only Properties, einer Methode und einem Event:

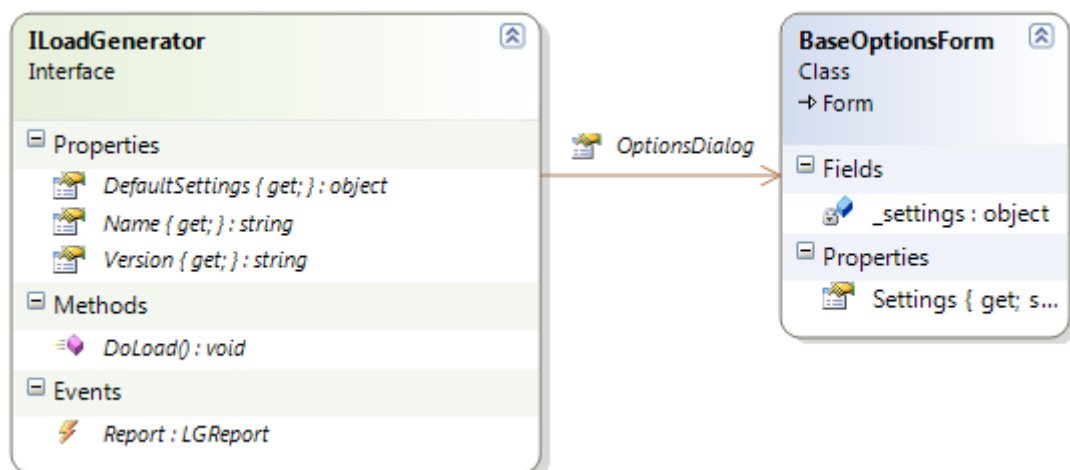


Abbildung 29: ILoadGenerator Schnittstelle - Wird von allen Surfers implementiert

- **DefaultSettings**
Gibt ein Objekt zurück, welches die Standardeinstellungen beinhaltet. Jeder LoadGenerator Implementation hat eine eigene Settings Klasse, welche als object im Konstruktor übergeben wird.
- **Name**
Der Name des Surfertypen. Er muss derselbe sein, wie die Klasse.
- **Version**
Versionsnummer
- **DoLoad()**
Wenn die LoadGenerator Anwendung das Startsignal vom Control-Center empfängt, führt es diese Methode aus. Sie erzeugt die Last gegen das MPP.

- **Report**
Wenn eine Nachricht an das Control-Center geschickt werden soll, kann dieses Event gefeuert werden. Die LoadGenerator Anwendung fängt dieses ab und ist für die Weiterleitung über das Netzwerk zum Control-Center verantwortlich.
- **OptionsDialog**
Gibt ein Objekt vom Typ BaseOptionsForm zurück. Diese Klasse ist abgeleitet von der Framework Klasse Form. Dieses Property wird vom Control-Center verwendet, um einen Dialog mit den Einstellmöglichkeiten anzuzeigen. Die ILoadGenerator Implementierung gibt eine Instanz einer von BaseOptionsForm abgeleiteten Klasse zurück. Dies ist eine Formulare Klasse, welche alle Einstellmöglichkeiten bereitstellt. Die BaseOptionsForm Klasse dient lediglich als Schnittstelle.

1.2 Contracts

Wenn eine neue Klasse implementiert wird, welche das ILoadGenerator Interface implementiert, müssen folgende Contracts eingehalten werden, um einen reibungslosen Ablauf zu garantieren:

- **DoLoad()**
Im Konstruktor werden die Anzahl Runs definiert. Diese Methode muss genau so viele Runs durchführen und am Ende von jedem Run das Report Event mit dem Ergebnis aufrufen. Falls Fehler bei der Übertragung der Nachricht entstehen, wird dies durch die LoadGenerator Applikation entsprechend geloggt.
- **Name**
Die Implementation des dynamischen Ladens von Klassen, welche das Interface ILoadGenerator implementieren, setzt voraus, dass dieses Property gleich wie der Name der Klasse ist.
- **Report**
Dieser Event muss immer überprüft werden, ob er null ist, bevor er aufgerufen wird.

2 Implementation

2.1 HTTP Request / Parsing von HTML Seiten

Ziel ist es, die Surfer möglichst echttheitsgetrau zu realisieren. Bei den Surfern die eine Website besuchen ist dies problematisch da kein Web Browser zur Verfügung steht, welcher den HTML Code parst und alle verlinkten Ressourcen herunterlädt.

Mittels den WebRequests, welche vom .NET-Framework zur Verfügung gestellt werden, wird ein Http Request auf die gewünschte Seite hergestellt. Durch diesen Request konnte der ganze HTML Quellcode von der Seite geholt werden und als String gespeichert werden. Mittels Regex wurde dieser String nach Verweisen geparkt und die Resultate in eine Liste gespeichert. Diese Liste wurde dann gefiltert um die Ressourcen einer Seite herauszukriegen um diese herunterzuladen. Als Ressourcen wurden Javascript-, CSS- und Bild – Dateien ausgewählt. Zudem braucht es noch Funktionen, um diese Verweise korrekt umzuwandeln. Das heisst, Relative Pfade mussten in Absolute Pfade umgewandelt werden. Da der Schwerpunkt von diesem Projekt nicht beim HTML Parsing liegt, wurde absichtlich darauf verzichtet, Ressourcen z.B. aus einer CSS Datei zu parsen.

Diese Funktionen können für jeden Surfer gebraucht werden welcher Websites besucht, und daher entstand die Web-Klasse welche die Funktionen zur Verfügung stellt.

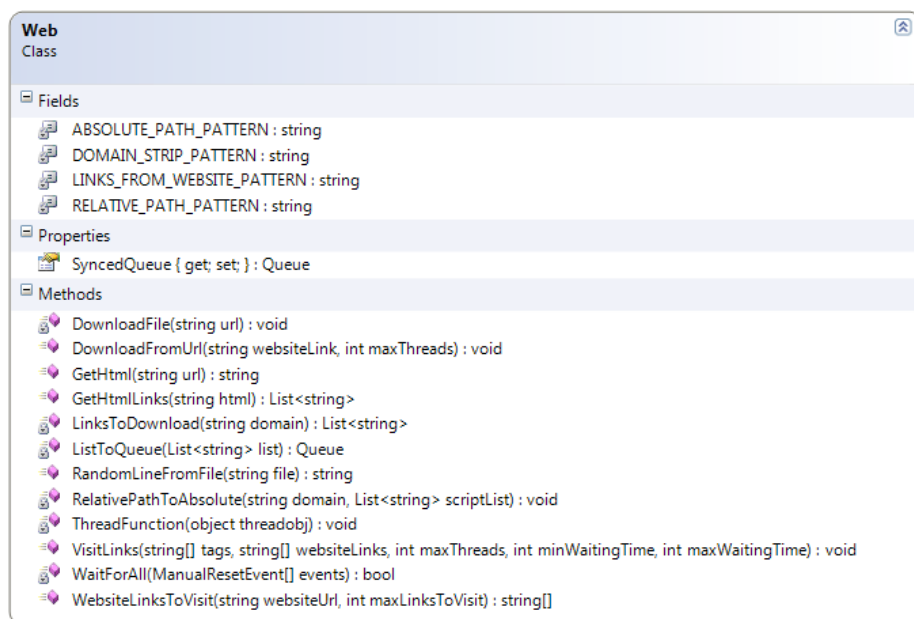


Abbildung 30: Die wichtigsten Methoden der Web-Klasse

Diese Klasse bietet statische Methoden, welche die Surfer bei Gebrauch benutzen können, um den ganzen Inhalt einer URL herunterzuladen. Die Anzahl der Threads, welche asynchron die einzelnen Ressourcen herunterladen ist variabel und je nach Wunsch einstellbar.

2.2 Umsetzung der Surfers

2.2.1 Googler

Mit der vorhandenen Web-Klasse ist ein Hauptproblem gelöst, um den Googler zu Implementieren. Hauptsächlich geht es nur noch um den sequentiellen Ablauf.. Zuerst wird aus einer Textdatei mit Suchwörtern zufällig eines ausgewählt. Mit diesem Suchwort wird ein Webrequest ausgeführt, um Suchresultate zu erhalten. Der HTML Code der Suchresultate wird mithilfe der Webklasse in ein Domain Object Model umgewandelt. In diesem Baum können anschliessend problemlos die Links entnommen werden.

Der Googler bietet Einstellungen welche vorgenommen werden können um das Verhalten den Bedürfnissen anzupassen. Es können die Anzahl Suchvorgänge, die Anzahl überprüfter Suchresultate und deren tiefen frei konfiguriert werden. Die Liste der Wörter, nach denen gesucht wird, muss in Form einer Textdatei vorliegen.

Für die Webrequests kann die Anzahl verwendeter Threads konfiguriert werden.

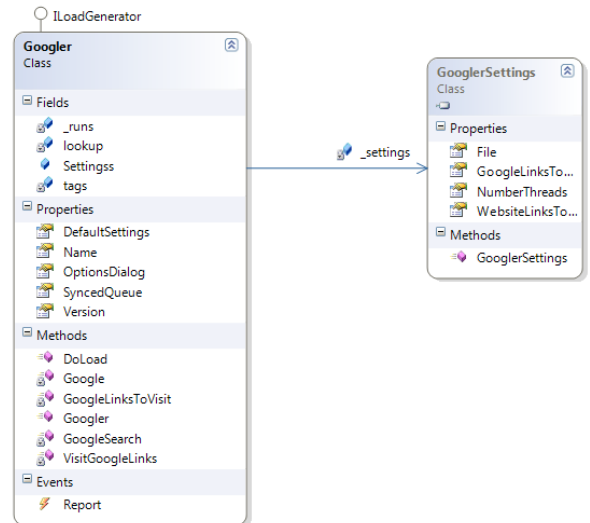


Abbildung 31: Klassendiagramm Googler

2.2.2 NewsReader

Der NewsReader hat grosse Ähnlichkeiten mit dem Googler. Er macht ebenfalls Gebrauch von der Web Hilfsklasse. Der grundlegende Unterschied zum Googler ist, dass sich der NewsReader bei jedem Seitenaufruf neu in das MPP einloggt. Der NewsReader bietet auch Einstellungen, welche das Verhalten des Surfers ändert. Es kann eingestellt werden, wie viele Newsites besucht und wie viele News gelesen werden sollen. Die verwendeten Newsseiten liegen wie die Wörter für den Googler in Form einer Textdatei vor.

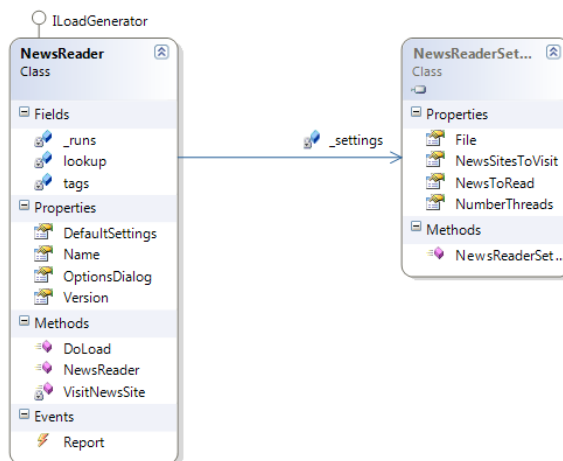


Abbildung 32: Klassendiagramm NewsReader

2.2.3 Downloader

Der Downloader ladet eine Ressource herunter, welche ihm durch die Optionen mittels einer URI mitgeteilt wird. Pro Run wird eine Datei heruntergeladen.

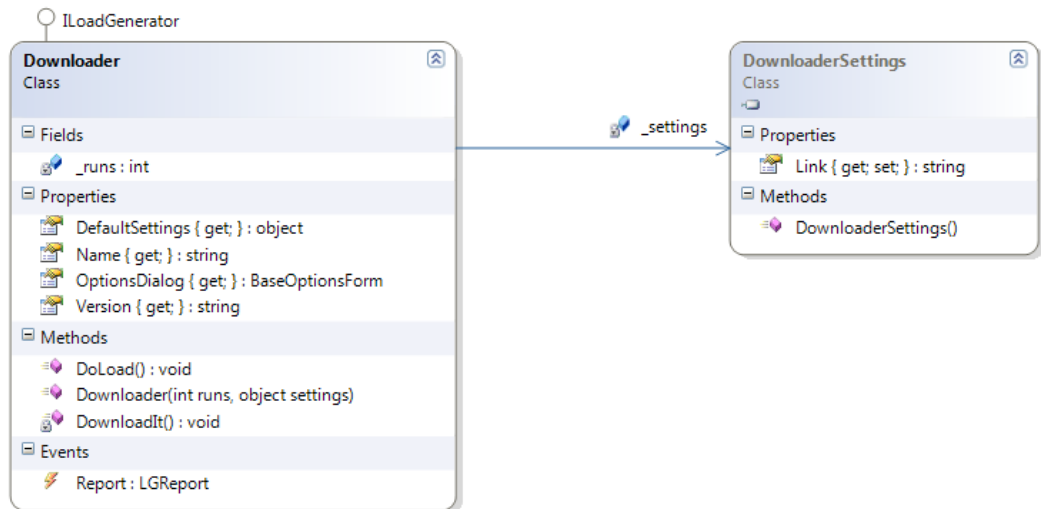


Abbildung 33: Klassendiagramm Downloader

2.2.4 MPPHammer

Beim MPPHammer geht es darum, ohne sich beim MPP einzuloggen, Webrequests abzuschicken. Dabei wird er vom MPP jedesmal auf die Landing Page umgeleitet. Die Seiten, welche er anfordert, werden in einer Textdatei gespeichert.

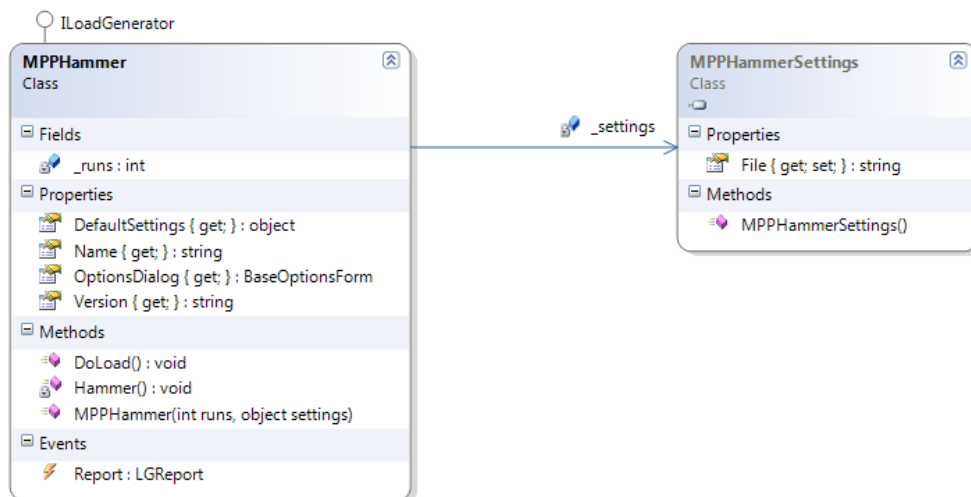
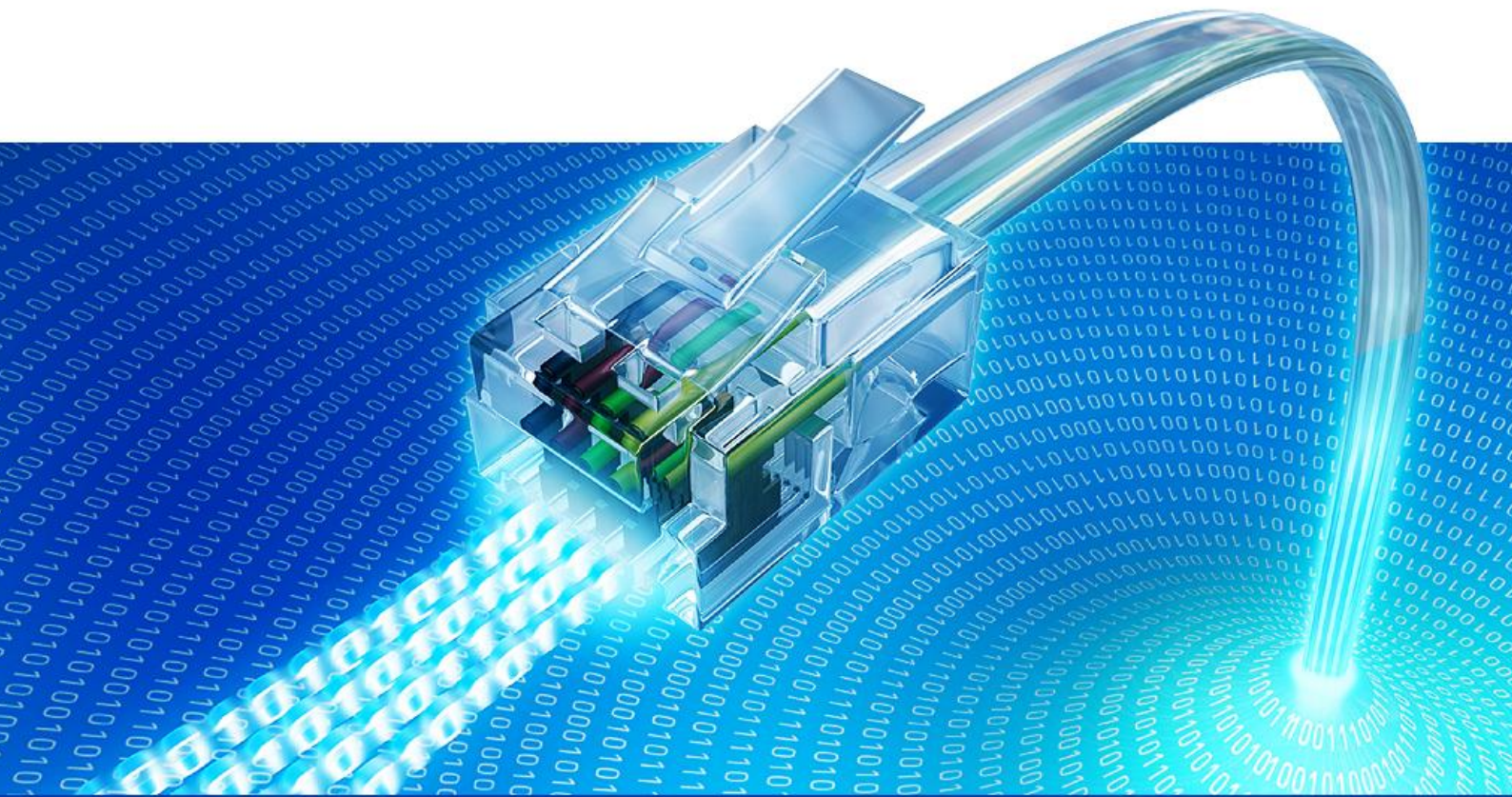


Abbildung 34: Klassendiagramm MPPHammer



12. Testdokumentation

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Unit Tests

Da das Domain-Modell nicht allzu gross ist, waren nicht sehr viele Unit Tests nötig. Es wurde viel Architektur-Code sowie Web-Klassen geschrieben, welche schwierig mit Unit-Tests zu testen sind. Denn dazu müssten diverse Elemente wie zum Beispiel das MPP oder ein Webserver „gemockt“ werden. Da dies zu viel Aufwand für den erhaltenen Nutzen sind und die 14 Wochen dafür einfach nicht ausgereicht hätten, wurde auf dies verzichtet. Stattdessen werden diese Klassen beim Systemtest getestet.

In der Visual Studio Solution ist ein UnitTest-Projekt zu finden, welches folgende UnitTests beinhaltet. Diese liefen bei der Abgabe alle 100% erfolgreich durch.

Test	Beschreibung
MacTest	Überprüft ob der MAC-Adressen Generator richtige MAC Adressen generiert ab einem bestimmten Offset.
NextNodeTest	Überprüft ob die Loadbalancing Strategie, für die Verteilung der VMs auf die Nodes richtig funktioniert.

2 Systemtest

Der Systemtest des Prototypen wurde in der Woche 12 im Praktikumsraum durchgeführt.

2.1 Durchlauf 1

Beim ersten Durchlauf musste spontan auf Ubuntu auf den Nodes gewechselt werden, da Debian auf dem neuen Rechner nicht lief. Bei der Entwicklung wurde ausschliesslich Debian Lenny eingesetzt, welches zwar ähnlich aber nicht ganz dasselbe ist. Die VMRemoteControl Applikation funktionierte nicht auf Anhieb, weswegen der Test abgebrochen wurde. Folgende Punkte mussten verbessert werden:

	Lösung
ifup/ifdown Script von KVM hat auf Ubuntu einen anderen Pfad	Die Scripte werden in das Verzeichnis /home/captive reingenommen und beim Starten der virtuellen Maschinen explizit angegeben.
WinForms2.0 Library fehlte	Aufgrund eines Fehler im Installationsscript fehlte nach der Installation das package libmono-WinForms2.0-cil. => wird hinzugefügt
sudoers	Die sudoers Datei schaut einwenig anders auf Ubuntu als auf Debian aus. Das Installationsscript muss dementsprechend angepasst werden.
KVM verteilte IPs, obwohl die	Dieses Problem war auf den Fehler mit den

IPs vom MPP DHCP erwartet wurden	Pfaden der ifup/ifdown Scripte zurückzuführen.
-----------------------------------------	------------------------------------------------

2.2 Durchlauf 2

Nachdem die Installation Reibungslos funktionierte, wurden folgende Use Cases getestet:

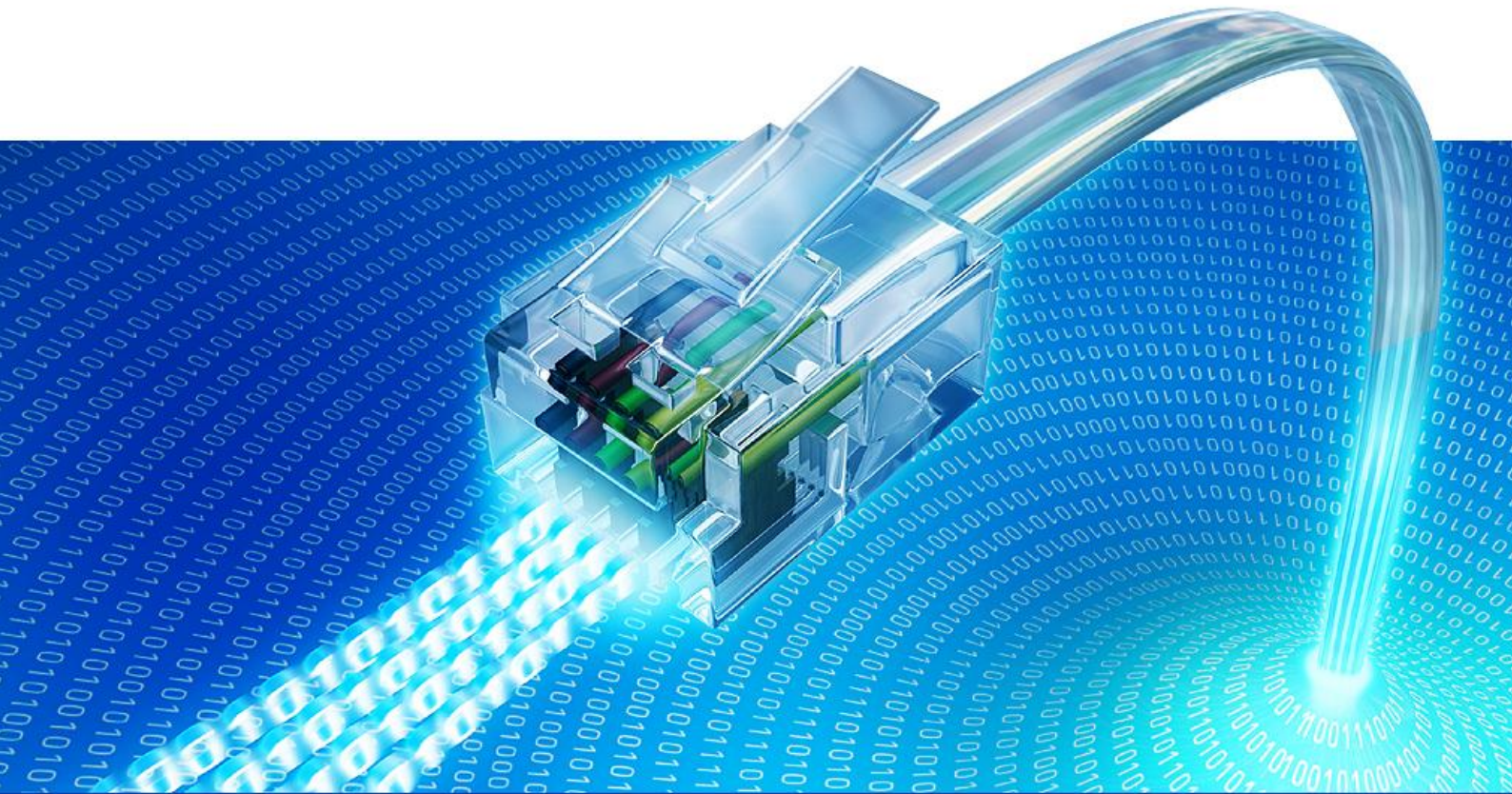
Use Case	Beschreibung	Status
Szenarios verwalten	Im Control-Center können Szenarien unter verschiedenen Namen gespeichert werden, damit sie beim nächsten Mal wiederverwendbar sind.	OK
Szenarios starten/stoppen	Nach der Konfiguration des Szenarios wird es gestartet. Dabei werden die konfigurierten Clients auf die Computer verteilt.	OK

Grundsätzlich lief der Test relativ gut. Die Architektur funktionierte auch mit 24 Computer (beim Entwickeln wurden nur 5 Nodes eingesetzt) und skalierte gut. Der Prototyp zeigt, dass es möglich ist, tausende Clients zu simulieren.

Obwohl der Prototyp funktionierte, traten Probleme auf, welche weiter verfolgt werden müssen:

Problem	Beschreibung
Timeouts	Die Surfers wurden mit relativ kleinen Timeouts implementiert. Die sind in der Höhe von einer Sekunde bis ca. 5 Sekunden. Diese Werte sind deutlich zu klein, da das MPP mit vielen Clients, die gleichzeitig kommen doch seine Zeit benötigt, um zu antworten.
Gleichzeitiges Starten aller Clients	Den Anforderungen entsprechend wurde das Starten eines Szenarios so implementiert, dass alle Surfers per Multicast signalisiert werden zu starten, damit der Start aller Clients möglichst zeitgleich stattfindet. Im praktischen Test hat dies gezeigt, dass das MPP bei 1 600 Clients auf einmal extrem ausgelastet wird und plötzlich eine Zeit lang nicht mehr antwortet. Ein Lösungsvorschlag für dieses Problem wäre, per Multicast in der Startnachricht eine Range mitzugeben, in welcher jeder Surfer eine Zufallszahl zieht und diese Zeit vor dem Start abwartet.
Konfigurationsmöglichkeiten	Eigenschaften wie der verwendete Login beim

	MPP von den Surfers sind Hardcoded und somit nicht konfigurierbar. Für den produktiven Einsatz müsste diese Einstellung konfigurierbar gemacht werden.
Login auf der MPP Seite	Die MPP Landingpage muss von den Surfers geparsed werden, damit das Formular zum einloggen ausgefüllt werden kann. Eine gewisse Toleranz lässt der verwendete Regex zu, allerdings wenn die Seite zu sehr abgeändert wird, werden die Felder nicht mehr erkannt.
AGB Checkbox	Bei dem MPP, welches wir für die Entwicklung verwendeten, war keine Checkbox auf der Landingpage, um die AGB's zu akzeptieren. Deshalb wurde bis zum Test diese Checkbox nicht beachtet. Dies wurde aber vor Ort noch verbessert.
Wie ausloggen wenn nie eingeloggt wurde?	Wenn eine VM eingeloggt ist, weil bereits eine Session offen ist, konnte nicht mehr ausgeloggt werden. Der Surfer kennt dazu die Landingpage nicht, denn diese kennt er nur durch den Redirect, wenn er eine Seite besuchen will und noch nicht eingeloggt ist. Eine Lösung wäre direkt das Gateway „anzusurfen“.



13. Anhang

Captive Portal Load Generator

Semesterarbeit HS 2009

Technische Hochschule Rapperswil

1 Installation VNUML unter Linux

Die Anleitung zeigt die Installation der Standard VNUML Umgebung. Ziel ist es, dass die Beispiele im example Ordner von VNUML lauffähig sind.

1.1 Grundinstallation

```
echo "deb http://jungla.dit.upm.es/~vnuml/debian binary/" >>
/etc/apt/sources.list
```

VNUML Source apt bekannt machen

```
apt-get update
```

Packagelist aktualisieren

```
apt-get install vnuml
```

VNUML installieren

```
apt-get install vlan xterm bridge-utils screen
```

Diese zusätzlichen Pakete werden benötigt

1.2 Installation des Kernels

apt-get install linux-um

Installiert den aktuellen VNUML Kernel nach `/usr/share/vnuml/kernels/`

```
cd /usr/share/vnuml/kernels
ln -s <kernelname> ./linux
```

Die Examples benötigen diese symbolische Verknüpfung, um korrekt zu funktionieren.

1.3 Installation des Dateisystems

```
curl, wget, head, grep, bunzip2, mktemp, mount, /usr/bin/time,
md5sum, perl
```

Diese Befehle müssen verfügbar sein, um das Dateisystem zu installieren.

```
wget http://jungla.dit.upm.es/~vnuml/download/scripts/root-fs-
installer
```

Perl Script herunterladen

```
perl root-fs-installer 0.5.0
```

Dateisystem automatisch herunterladen und installieren

```
cd /usr/share/vnuml/filesystems
ln -s <name> ./root_fs_tutorial
```

Symbolische Verknüpfung, damit die Beispiele funktionieren

1.4 Beispiel einer Ausführung

```
vnumlparser.pl -t /usr/share/vnuml/examples/simple.xml -v
```

Beispielsszenario starten.

```
vnumlparser.pl -x seq_name@/usr/share/vnuml/examples/simple.xml
```

Pro VM Eintrag in der XML Datei können `<exec>` Tags definiert werden. Diese enthalten als Attribute den Befehl und den Benutzer, welcher den Befehl ausführt.

Durch den Aufruf des Parsers mit dem `-x` Parameter, werden diese Befehle ausgeführt (dabei muss das Szenario bereits mit `-t` gestartet worden sein).

```
<exec seq="seq_name" type="verbatim">  
  nohup /usr/bin/hello </dev/null >/dev/null 2>&1 &  
</exec>
```

```
vnumlparser.pl -d /usr/share/vnuml/examples/simple.xml
```

Beendet das Szenario. Es werden alle VMs heruntergefahren.

2 Linux Kernel kompilieren

ACHTUNG: Wenn der Kernel mit VMWare verwendet wird, sollten IDE Festplatten verwendet werden. Der Treiber für die SCSI Festplatten hatte bei unseren Versuchen nie funktioniert.

2.1 Konfiguration

```
wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-  
2.6.31.1.tar.bz2
```

Kernel herunterladen

```
tar -xvjf linux-2.6.31.1.tar.bz2
```

Kernel entpacken

```
cd linux-2.6.31.1
```

In den Ordner des entpackten Kernel wechseln

```
make menuconfig
```

Wenn der `make` Befehl abgeschlossen ist, wird ein Menu angezeigt, in dem der Kernel konfiguriert werden kann.

```
make
```

Nachdem im Menu der Kernel konfiguriert wurde, kann er mit dem Befehl `make` kompiliert werden.


```
cp arch/x86/boot/bzImage /boot/vmlinuz-tutorial
```

Anschliessend den kompilierten Kernel in den /boot Ordner kopieren.

2.2 Kernel im Bootmanager (LILO) eintragen

```
vi /etc/lilo.conf
```

Ganz am Ende kann nun ein neuer Eintrag erstellt werden:

```
Image=/boot/vmlinuz-tutorial  
label=KernelTutorial  
read-only
```

Anschliessend müssen die Änderungen übernommen werden:

```
lilo
```

Beim booten muss nun die Shift Taste gedrückt werden, damit nicht automatisch vom alten Kernel gebootet wird. Im Menu von Lilo kann nun „Kernel Tutorial“ ausgewählt werden.

3 Installation von Xen unter Linux

ACHTUNG: Damit Xen funktioniert, muss der GRUB Bootloader installiert sein.

3.1 Basis Installation

```
apt-get update
```

Packagelist aktualisieren

```
apt-get install xen-hypervisor-3.2-1-i386
```

Der Hypervisor ist der "core" von Xen. Dies wird am Anfang durch den Bootloader gebootet, um im System die Möglichkeit zu bekommen, die Virtuellen Maschinen zu starten und verwalten.

```
apt-get install xen-tools
```

Zusätzliche Tools um die Xen VMs zu verwalten.

```
apt-get install linux-image-2.6.26-2-xen-686
```

Installiert den Xen Kernel um Xen-Hypervisor zu booten.

```
shutdown -r now
```

Systemreboot, beim Booten den neu installierten Kernel auswählen

```
xend start
```

Xen Daemon aufstarten.

3.2 Konfiguration einer Virtuellen Maschine mittels xen-tools

```
nano /etc/xen-tools/xen-tools.conf
```

Konfigurationsdatei um Basiseinstellungen vorzunehmen.

```
dir = /home/xen
install-method = debootstrap
size = 4GB
memory = 36MB
swap = 65MB
noswap = 1 //um swapping auszustellen
fs = ext3
dist = lenny
image = sparse
gateway = 192.168.1.1
netmask = 255.255.255.0
broadcast = 192.168.1.255
mirror = http://ftp.ch.debian.org/debian/
```

Werte, welche im Konfigurationsfile angepasst werden sollten.

```
xm-create-image --hostname=virtualMaschine -ip 192.168.1.99
```

Alle benötigten Files für die Virtuelle Maschine durch die xen-tools werden erzeugt.

```
xm create virtualMaschine.cfg
```

Virtuelle Maschine wird gestartet

```
xm list
```

Übersicht über die Virtuellen Maschinen auf dem Hostsystem

```
xm console virtualMaschine
```

In die Konsole der gewünschten Virtuellen Maschine wechseln.

3.3 NAT-Konfiguration

```
nano /etc/xen/xend-config.sxp
```

Konfigurationsdatei von Xen öffnen

```
(network-script network-nat)
(vif-script vif-nat)
```

Diese Konfigurationen auskommentieren und die default Dummy Konfigurationen löschen. Diese enthalten „dummy“ in ihrem Namen.

3.4 GRUB-Bootloader Eintrag (Optional)

```
title Xen 3.2-1-i386 / Debian GNU/Linux, kernel 2.6.26-2-xen-686
root (hd0,2)
kernel /boot/xen-3.2-1-i386.gz
module /boot/vmlinuz-2.6.26-2-xen-686 root=/dev/sda3 ro console=tty0
module /boot/initrd.img-2.6.26-2-xen-686
```

Einträge für GRUB in /boot/grub/menu.lst

4 KVM-Installation

Die Anleitung zeigt die Installation von KVM. Ziel ist es zu zeigen, wie ein Base-Image erstellt wird, wie Klone erzeugt werden und dass die Klone per DHCP eine IP bekommen.

4.1 Basis Installation

```
apt-get install kvm
```

Damit wird die KVM Umgebung installiert.

```
kvm-img create -f qcow2 image.img 8G
```

Mit diesem Befehl wird das Dateisystem für eine Virtuelle Maschine erstellt.

```
kvm -hda image.img -m 51 -cdrom DebianInstaller.iso -boot d
```

Hier wird eine Virtuelle Maschine gestartet mit dem vorher erstellten Dateisystem gemounted als hda. Als zweites Laufwerk wird ein CDROM gemounted mit dem DebianInstaller.iso. Die Option -boot d lässt die VM direkt von dem CDROM booten.

```
kvm -hda image.img -m 51
```

Mit diesem Befehl wird die erstellte virtuelle Maschine gestartet, auf welchem zuvor das Debian installiert wurde.

4.2 Netzwerkeinrichtung

Damit die virtuellen Maschinen Zugriff auf das Netzwerk haben, muss auf dem Host-System eine Bridge eingerichtet werden.

```
apt-get install uml-utilities bridge-utils
nano /etc/network/interfaces
```

Die interfaces Datei muss wie folgt aussehen:

```
iface eth0 inet manual
auto br0
iface br0 inet dhcp
    bridge_ports eth0
    bridge_stp off
    bridge_fd 0
    bridge_maxwait 0
```

Damit wird bei einem Neustart vom System automatisch die Bridge br0 angelegt.

```
shutdown -r now
```

Systemneustart um die Änderungen zu übernehmen

```
nano /etc/kvm/kvm-ifup
```

Das Skript, welches automatisch aufgerufen wird von KVM, wenn die Netzwerkparameter übergeben werden.

```
#!/bin/sh
set -x

switch=br0

if [ -n "$1" ];then
    /usr/bin/sudo /usr/sbin/tunctl -u `whoami` -t $1
    /usr/bin/sudo /sbin/ip link set $1 up
    sleep 0.5s
    /usr/bin/sudo /usr/sbin/brctl addif $switch $1
    exit 0
else
    echo "Error: no interface specified"
    exit 1
fi
```

Mit diesem Skript werden dynamisch Tap-Devices angelegt.

4.3 Klone erstellen und starten

```
kvm-img create -b image.img -f qcow2 clon1.qcow2
```

Klon „clon1“ von image.img erstellen.

```
kvm -hda clon1.qcow2 -net nic,macaddr=DE:AD:BE:EF:12:34 -net tap -m 51
```

Der Klon wird gestartet und der Netzzugang wird automatisch erstellt.

5 Installationscripts

5.1 Node

Dieses Script erledigt die Grundkonfiguration einer Linux Debian Installation, damit die VMRemoteControl Applikation verwendet werden kann:

```
#!/bin/sh

if [[ $EUID -ne 0 ]]; then
    echo "This script must be run as root" 1>&2
    exit 1
fi

echo "#####BASIC SETUP#####\n"

apt-get update
apt-get install -y bridge-utils uml-utilities kvm mono-runtime libmono-corlib2.0-cil libmono-
winforms2.0-cil libmono-system-runtime2.0-cil sudo
useradd --create-home captive
mkdir /home/captive/bin
mkdir /home/captive/vms
mkdir /home/captive/vms/log
cd /home/captive/bin
wget ymyr.net/sa/get.sh
chmod +x get.sh
./get.sh

echo "finished BASIC SETUP\n"
echo "#####BRIDGE SETUP#####\n"
echo -n > /etc/network/interfaces
echo "#" >> /etc/network/interfaces
echo "auto lo" >> /etc/network/interfaces
```

```

echo "iface lo inet loopback" >> /etc/network/interfaces
echo "allow-hotplug eth0" >> /etc/network/interfaces
echo "iface eth0 inet manual" >> /etc/network/interfaces
echo "auto br0" >> /etc/network/interfaces
echo "iface br0 inet dhcp" >> /etc/network/interfaces
echo "
    bridge_ports eth0" >> /etc/network/interfaces
echo "
    bridge_stp off" >> /etc/network/interfaces
echo "
    bridge_fd 0" >> /etc/network/interfaces
echo "
    bridge_maxwait 0" >> /etc/network/interfaces

/etc/init.d/networking restart

echo "Restarted Networking, finished BRIDGE SETUP\n"

echo "#####KVM IFUP SCRIPT#####\n"
echo -n > /home/captive/kvm-ifup
echo "#!/bin/sh" >> /home/captive/kvm-ifup
echo "switch=br0" >> /home/captive/kvm-ifup
echo "if [ -n \"\$1\" ];then" >> /home/captive/kvm-ifup
echo "
    /usr/bin/sudo /usr/sbin/tunctl -u `whoami` -t \$1" >> /home/captive/kvm-ifup
echo "
    /usr/bin/sudo /sbin/ip link set \$1 up" >> /home/captive/kvm-ifup
echo "
    sleep 0.5s" >> /home/captive/kvm-ifup
echo "
    /usr/bin/sudo /usr/sbin/brctl addif \$switch \$1" >> /home/captive/kvm-ifup
echo "
    exit 0" >> /home/captive/kvm-ifup
echo "else" >> /home/captive/kvm-ifup
echo "
    exit 1" >> /home/captive/kvm-ifup
echo "fi" >> /home/captive/kvm-ifup

chmod +x /home/captive/kvm-ifup

echo "finished KVM IFUP SCRIPT\n"

echo "#####KVM IFDOWN SCRIPT#####\n"
echo -n > /home/captive/kvm-ifdown
echo "#!/bin/sh" >> /home/captive/kvm-ifdown
echo "if [ -n \"\$1\" ];then" >> /home/captive/kvm-ifdown

```

```
echo "          /usr/bin/sudo /usr/sbin/tunctl -d \$1" >> /home/captive/kvm-ifdown
echo "          sleep 0.5s" >> /home/captive/kvm-ifdown
echo "          exit 0" >> /home/captive/kvm-ifdown
echo "else" >> /home/captive/kvm-ifdown
echo "          exit 1" >> /home/captive/kvm-ifdown
echo "fi" >> /home/captive/kvm-ifdown

chmod +x /home/captive/kvm-ifdown

echo "finished KVM IFDOWN SCRIPT\n"

echo "#####SUDOERS SCRIPT#####\n"

echo "root      ALL=(ALL) ALL" >> /etc/sudoers
echo "captive   ALL=(ALL) /usr/sbin/tunctl /usr/sbin/brctl /sbin/ip" >> /etc/sudoers

echo "finished SUDOERS SCRIPT\n"
```

5.2 Base Image – Load Generator

Dieses Script erledigt die Grundkonfiguration des VM-Images:

6 get.sh

```
#!/bin/sh

cd /home/captive/bin

mono MPPLogin.exe

rm CapacityBar.dll
rm Captive.Common.dll
rm Captive.LoadGenerator.DemoSurfer.dll
rm Captive.LoadGenerator.DemoSurfer2.dll
rm Captive.LoadGenerator.Googler.dll
rm Captive.LoadGenerator.NewsReader.dll
rm Captive.LoadGenerator.Downloader.dll
rm Captive.LoadGenerator.WebSiteTrasher.dll
rm Captive.Virtualisation.KVMController.dll
rm Captive.LoadGenerator.MPPHammer.dll
rm controlcenter.exe
rm loadgenerator.exe
rm vmremotecontrol.exe
rm sites.txt
rm keywords.txt

echo "DELETED ALL FILES"

wget myr.net/sa/debug/CapacityBar.dll
wget myr.net/sa/debug/Captive.Common.dll
wget myr.net/sa/debug/Captive.LoadGenerator.DemoSurfer.dll
wget myr.net/sa/debug/Captive.LoadGenerator.DemoSurfer2.dll
```



```
wget ymyr.net/sa/debug/Captive.LoadGenerator.Googler.dll
wget ymyr.net/sa/debug/Captive.LoadGenerator.NewsReader.dll
wget ymyr.net/sa/debug/Captive.LoadGenerator.WebSiteTrasher.dll
wget ymyr.net/sa/debug/Captive.LoadGenerator.Downloader.dll
wget ymyr.net/sa/debug/Captive.LoadGenerator.MPPHammer.dll
wget ymyr.net/sa/debug/Captive.Virtualisation.KVMController.dll
wget ymyr.net/sa/debug/MPPLogin.exe
wget ymyr.net/sa/debug/controlcenter.exe
wget ymyr.net/sa/debug/loadgenerator.exe
wget ymyr.net/sa/debug/vmremotecontrol.exe
wget ymyr.net/sa/debug/sites.txt
wget ymyr.net/sa/debug/keywords.txt

echo "DOWNLOADED ALL FILES"

rm *.dll.*
rm *.exe.*
rm *.txt.*
```

7 Quickstarterguide

7.1 Übersicht

Das Control Center besteht aus einer Hauptsicht, welche alle Funktionen bereitstellt.

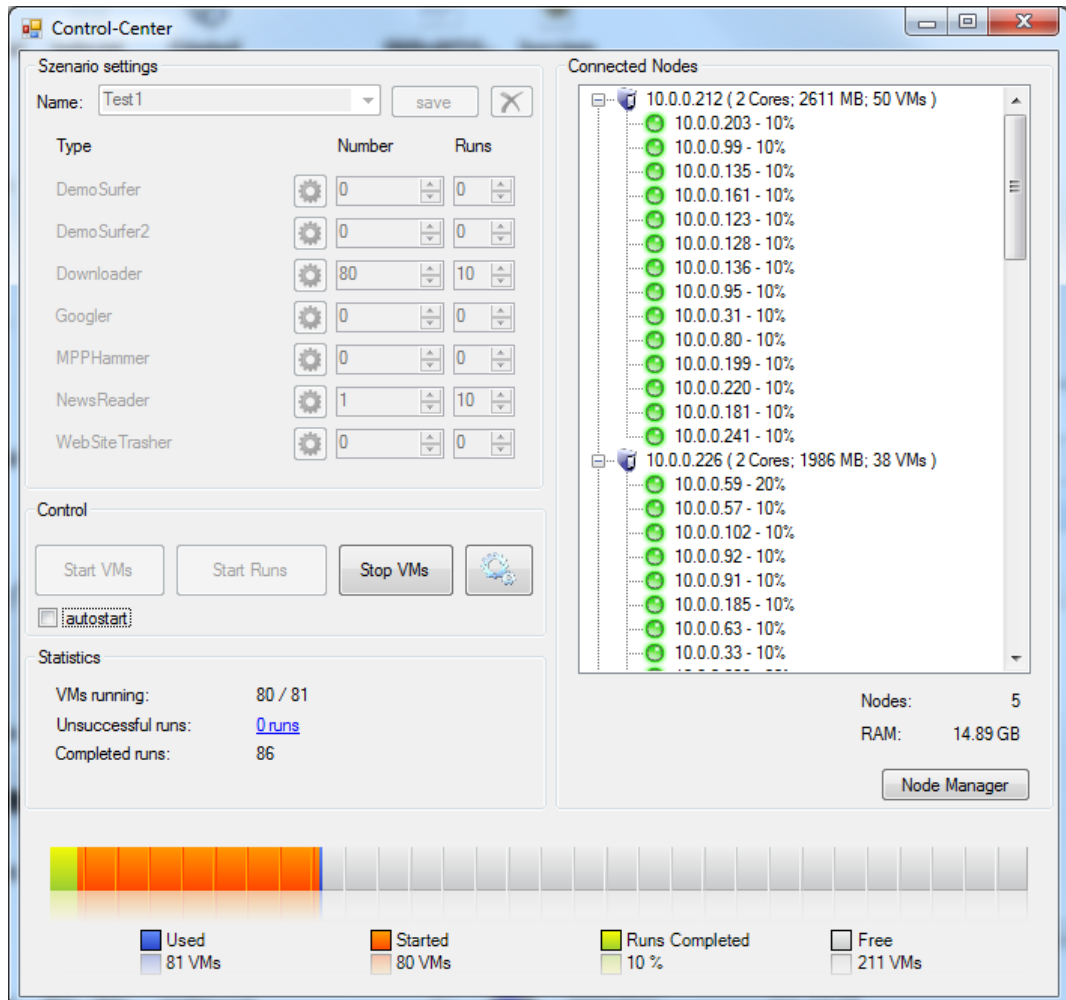


Abbildung 35: Control-Center GUI

7.1.1 Szenario Settings

In den Szenario Settings hat man die Möglichkeit, Szenarios zu erstellen. Diese kann man speichern, editieren und löschen.

- Type: Vorhandene Surfertypen
- Number: Anzahl Instanzen dieses Surfers
- Runs: Anzahl Durchläufe pro Instanz

7.1.2 Connected Nodes

Dies ist eine Übersicht, welche einen Überblick erschafft, über die vorhandenen Nodes und Ressourcen.

Jede Node zeigt in einer Baumansicht an, wie viele Virtuelle Maschinen gestartet sind. Mit einem Doppelklick auf die virtuelle Maschine erhält man weitere Informationen. Diese Infos enthalten den Surfer Typ, Anzahl erfolgreicher Durchläufe oder fehlgeschlagene Durchläufe.

7.1.3 Node Manager

Mithilfe des Node Managers werden die wartenden Nodes aktiviert. Diese sind in einer Auswahlliste zu sehen und können nach Belieben ausgewählt werden. Die ausgewählten Nodes werden in der Connected Nodes Übersicht angezeigt.

7.1.4 Control

Im Control steuert man die Szenarien und bietet noch ein Einstellungsmenü, indem man z.B. den eigenen SSH-Client einstellen kann.

Im Control sind die Hauptfunktionen des CLG zu finden. Zusätzlich hat das Control Center noch Einstellungen wie z.B. den eigenen SSH-Client einzustellen.

7.1.5 Statistics

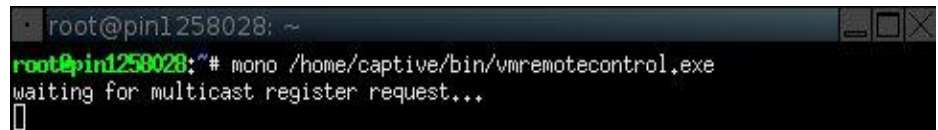
Eine kurze Zusammenfassung der ganzen Prozedur.

7.1.6 Capacity Bar

Eine Übersicht über den aktuellen Stand des Szenarios. Was gerade passiert und wie die Ressourcen eingeteilt sind.

7.2 Quickstart

- Auf den gewünschten Nodes das VMRemoteControl starten mittels „mono /home/captive/bin/vmremotecontrol.exe



```
root@pin1258028: ~  
root@pin1258028:~# mono /home/captive/bin/vmremotecontrol.exe  
waiting for multicast register request...  
█
```

Abbildung 36: Konsolenausgabe von VMRemoteControl.exe

- Control Center starten
- Im Control Center auf Node Manager klicken

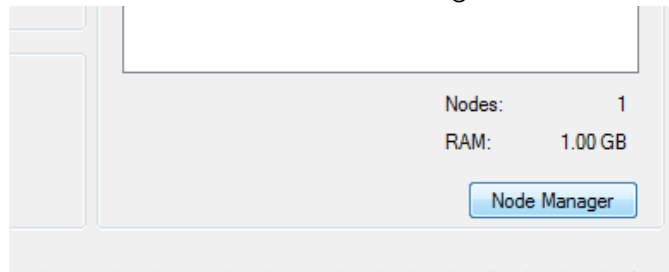


Abbildung 37: Node Manager

- Im neuen Fenster auf Start klicken

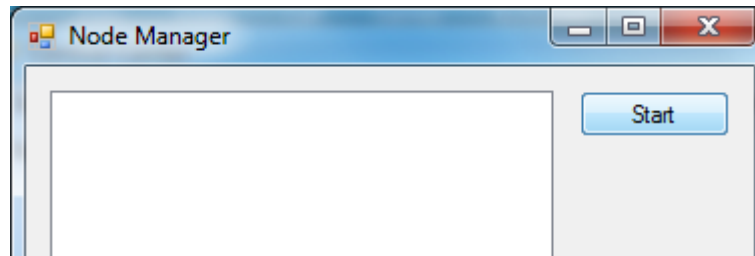


Abbildung 38: Nodes triggern

- Aus der Liste die gewünschten Nodes auswählen und OK klicken

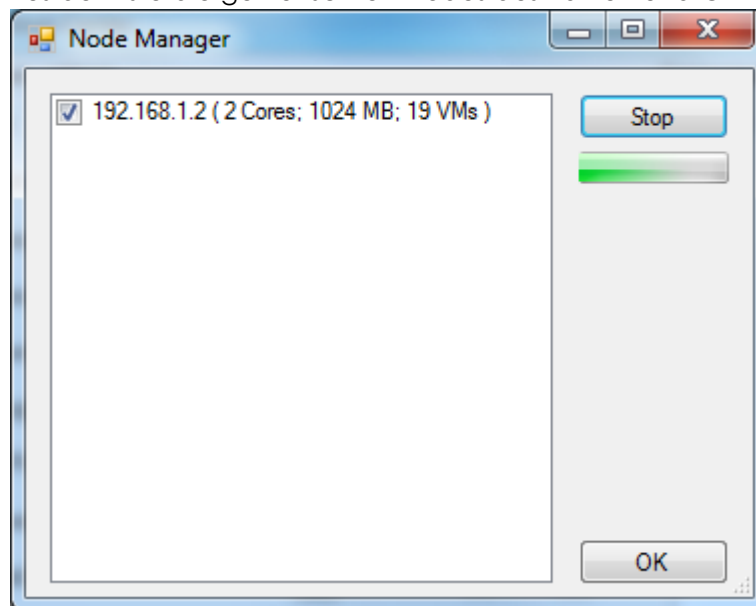


Abbildung 39: Registrierte Nodes werden aufgelistet

- Dem Szenario einen Namen geben

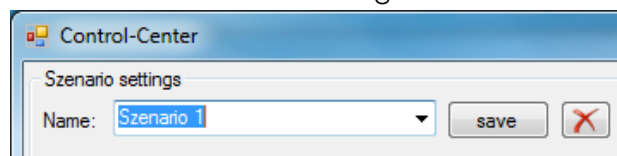


Abbildung 40: Szenarioname

- Surfer nach Gebrauch und vorhandenen Ressourcen verteilen

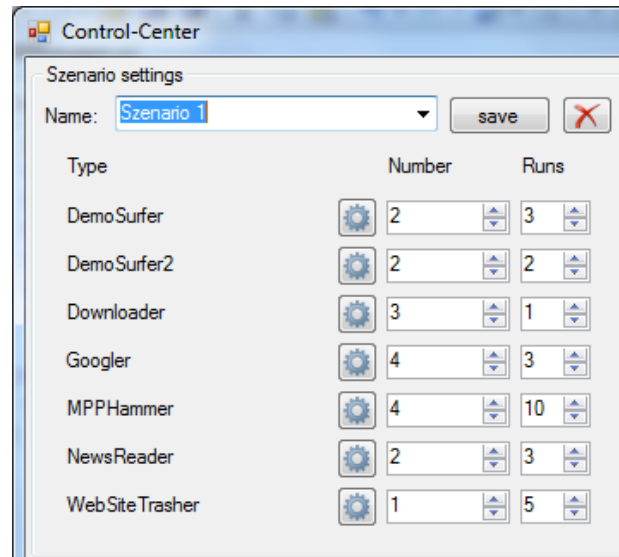


Abbildung 41: Einstellungen des Szenarios

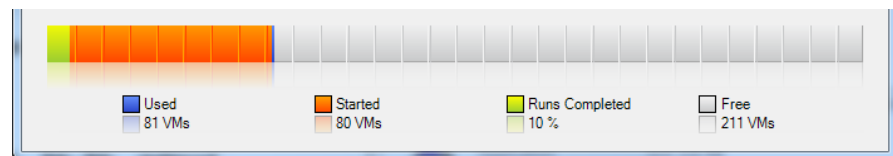


Abbildung 42: Zeigt die Kapazität und Fortschritt an

- Szenario speichern
- Auf „Start VMs“ klicken und warten bis der orange Balken sich füllt
- Wenn der blaue Balken vom Orangenen überdeckt wird, sind alle VMs gestartet. Über den Knopf „Start Runs“ kann das Szenario gestartet werden. Wenn eine Anzahl Runs definiert wurde füllt sich der orangene Balken.
- Szenario beendet

7.3 Weitere Funktionen

Jeder der Einzelnen Surfer hat eigene Einstellungen, welche über den Settings Knopf im GUI zu erreichen sind.



7.3.1 Downloader

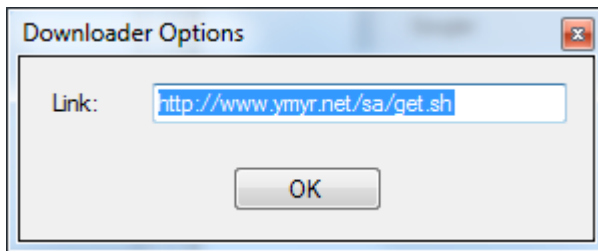


Abbildung 43: Settings Dialog Downloader

Der Downloader hat einzig eine Option. Es wird lediglich die URL zur Datei, welche zu herunterladen ist, eingefügt.

7.3.2 Googler

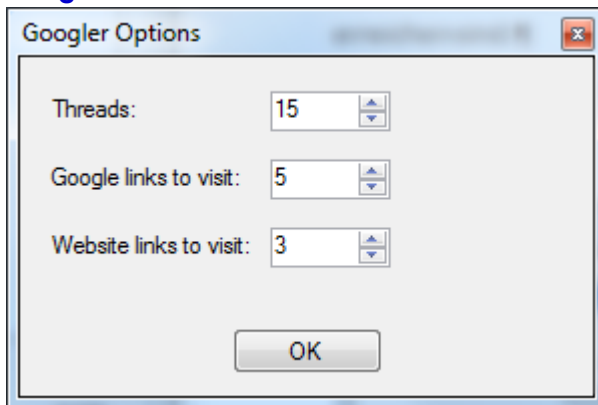


Abbildung 44: Settings Dialog Googler

Threads: Anzahl Threads welche zum Downloaden der Ressourcen gebraucht werden

Google Links: Wie viele Suchresultate besucht werden je Suche

Website Links: Wie tief auf der besuchten Website, aus der Suche, gesucht wird.

7.3.3 NewsReader

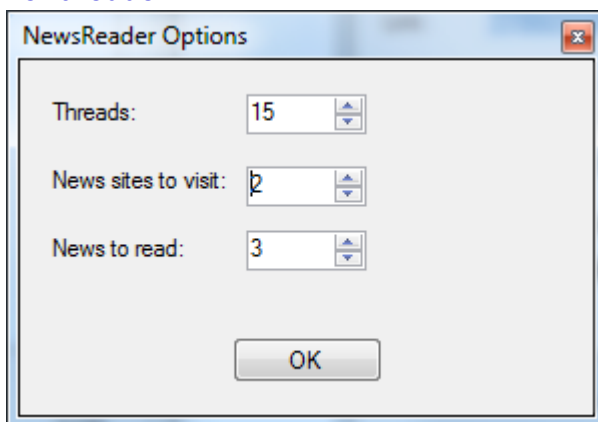


Abbildung 45: Settings Dialog NewsReader

Threads: Anzahl Threads welche zum Downloaden der Ressourcen gebraucht werden

News Links: Wie viele News Seiten besucht werden sollen
 News Reading: Wie viele News auf der besuchten Seite besucht werden sollen

7.3.4 WebsiteTrasher

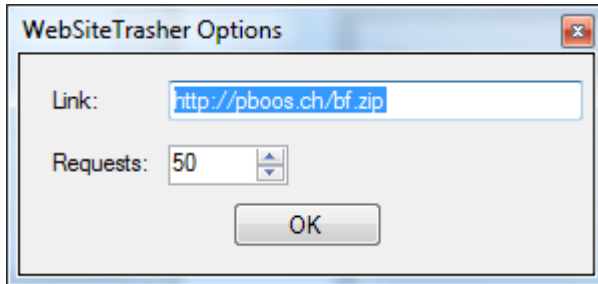


Abbildung 46: Settings Dialog WebSiteTrasher

Link: URL zur Datei, welche zu herunterladen ist, eingefügt
 Requests: Wie oft die zu herunterladende Datei asynchron heruntergeladen wird

7.3.5 Control Center Settings

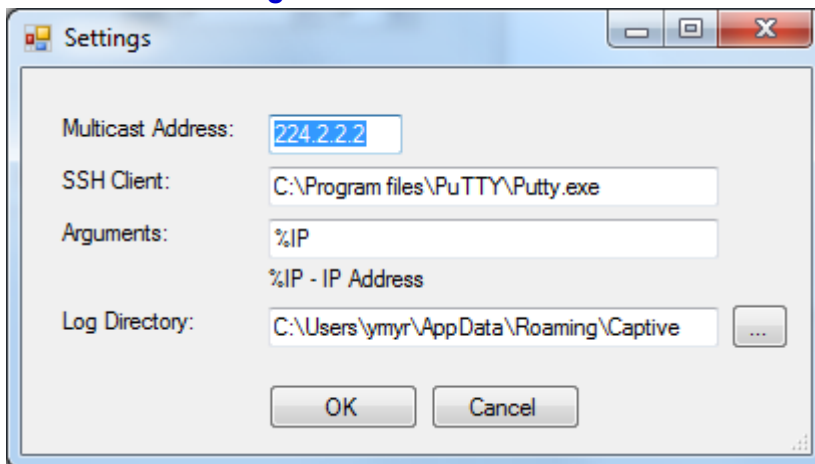
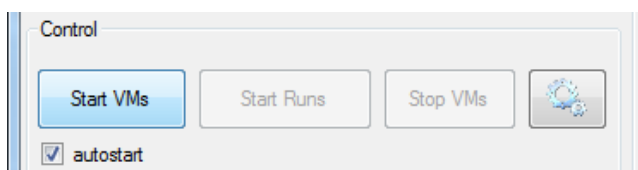


Abbildung 47: Control-Center Settings

SSH Client: Pfad zum SSH Client Programm
 Log Directory: Pfad für Logdatei

Zusätzlich zu diesen Einstellungen gibt es noch die Autostart Checkbox. Beim Aktivieren dieser Checkbox starten die Runs automatisch nachdem die VMs bereit sind.



7.3.6 Capacity Bar

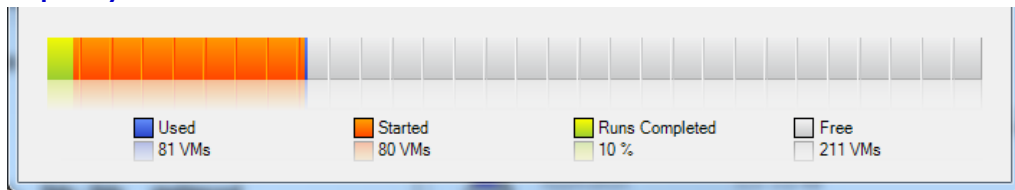
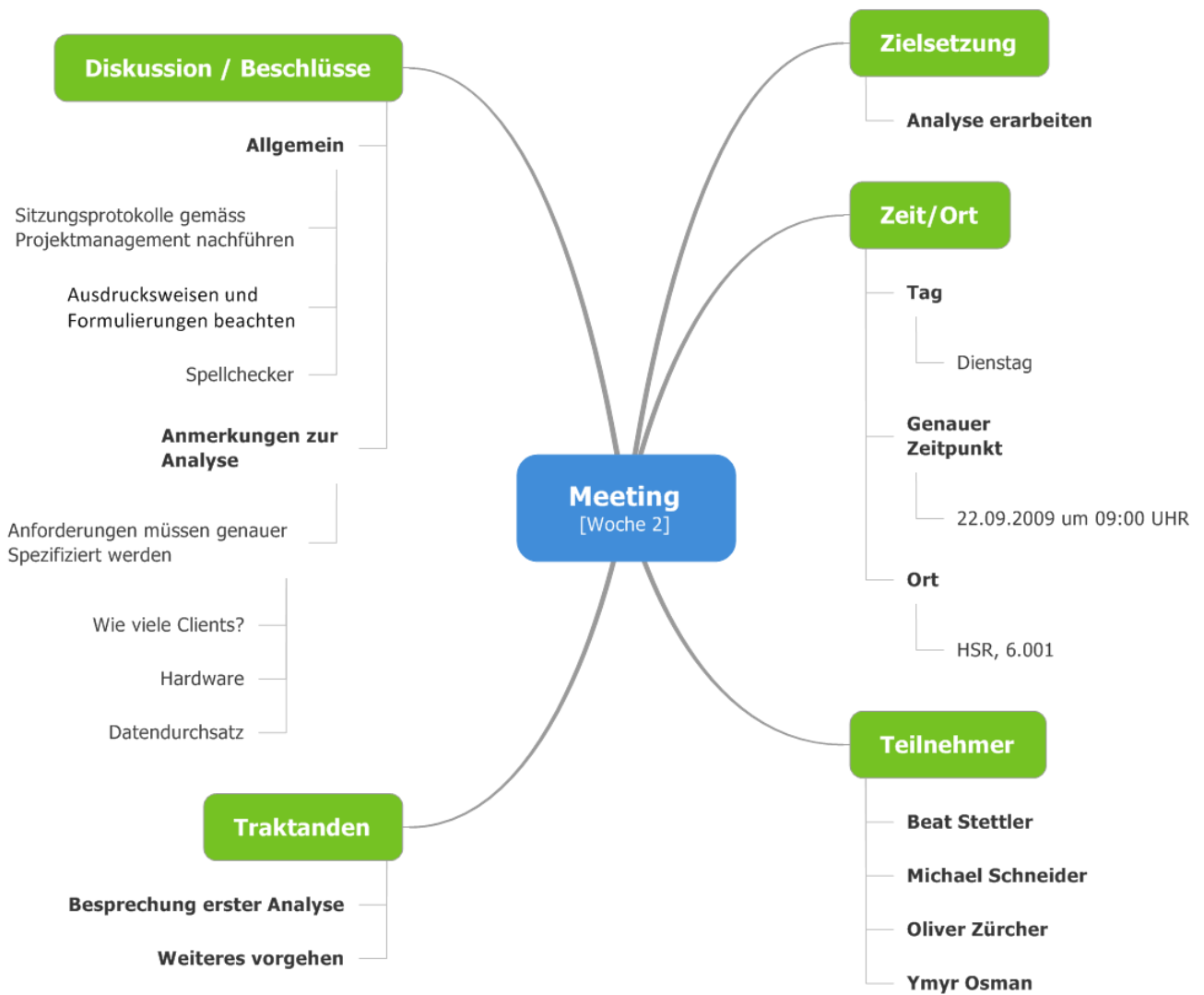


Abbildung 48: Capacity Bar

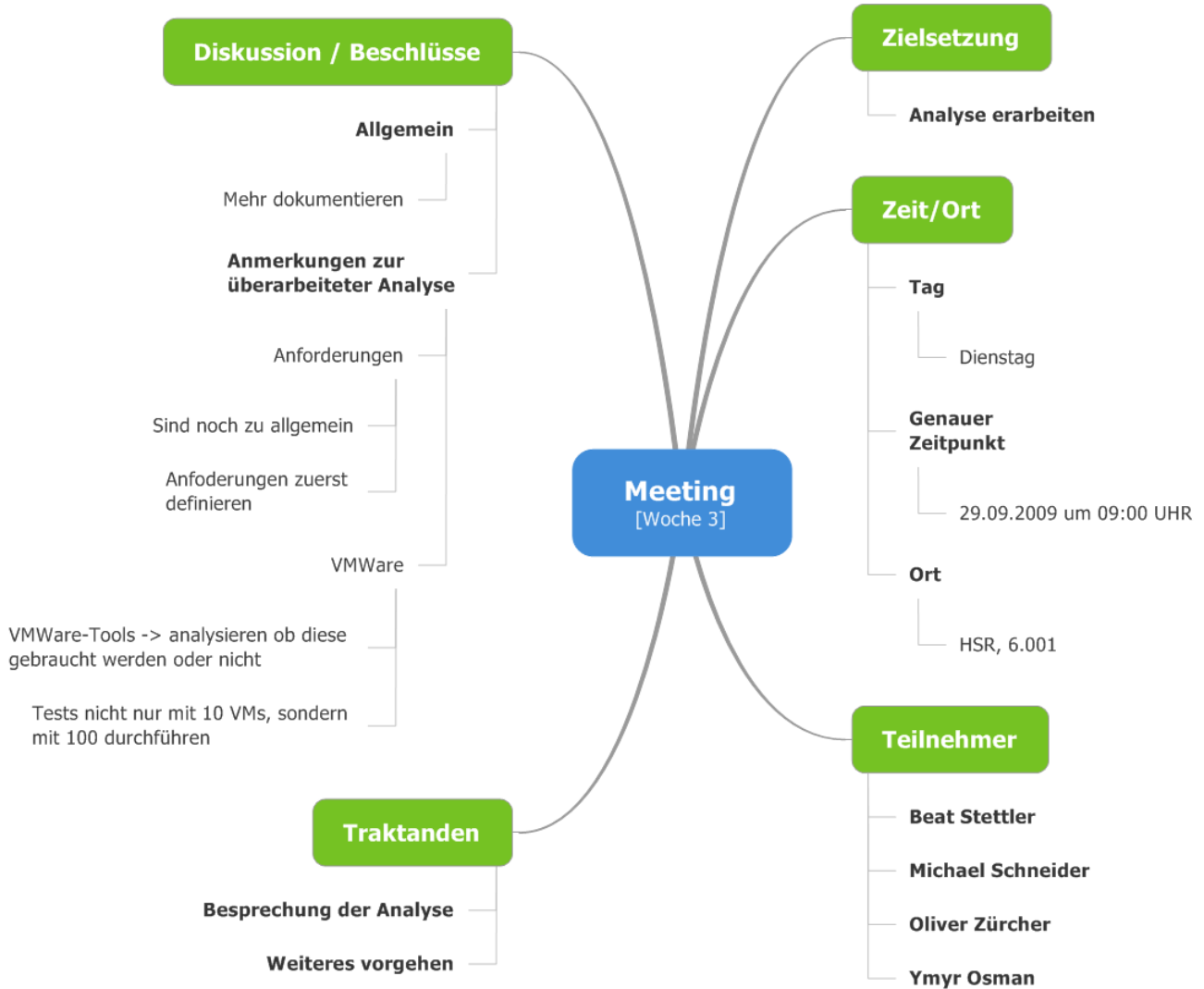
- Grau: Zeigt die maximal möglichen Ressourcen an Virtuellen Maschinen
- Blau: Zeigt die Anzahl der gebrauchten virtuellen Maschinen für das Szenario
- Orange: Zeigt an wie viele Virtuelle Maschinen bereit zum starten der Runs sind
- Geld: Zeigt den Fortschritt der Runs

8 Sitzungsprotokolle

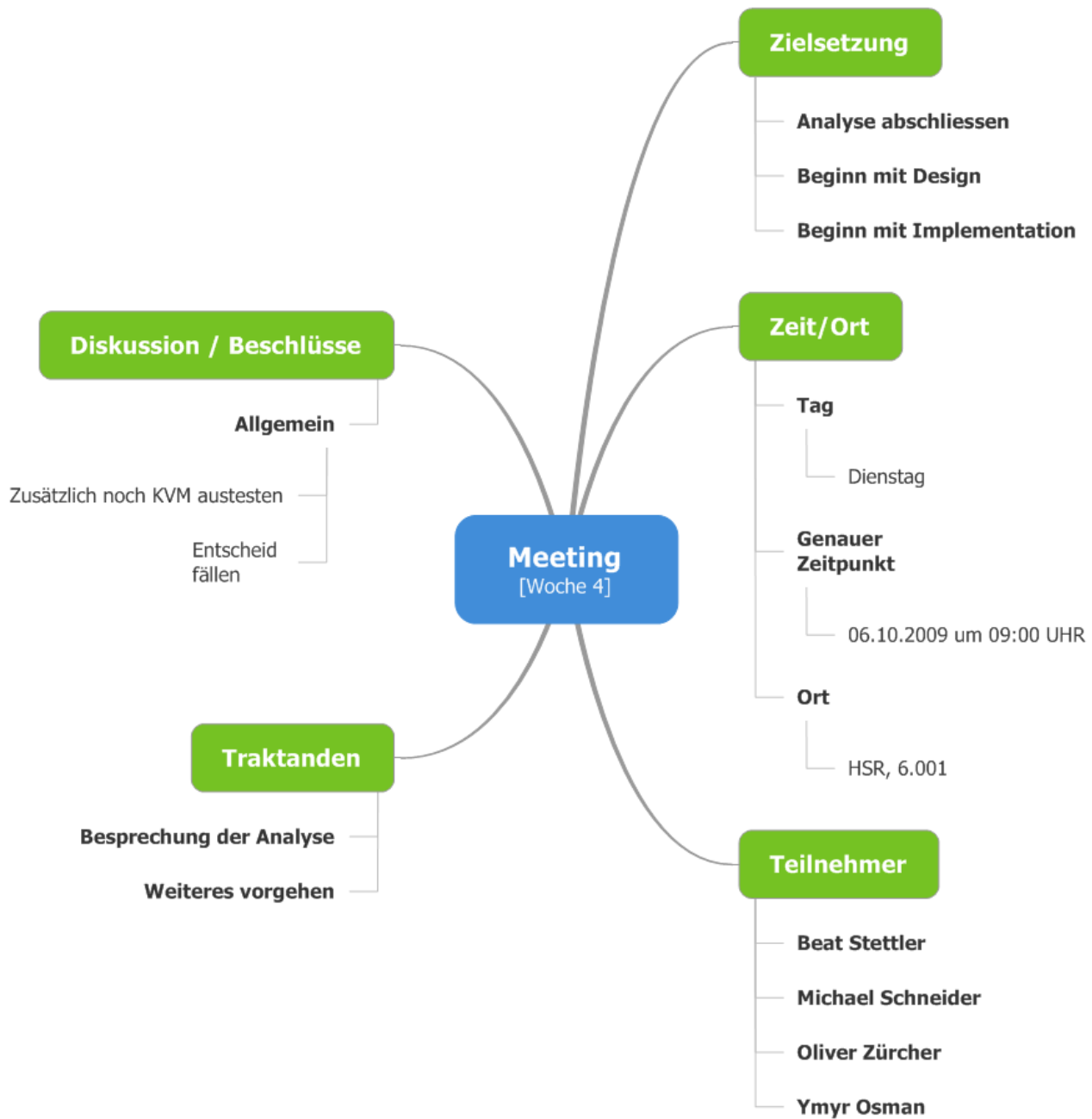
8.1 Woche 2



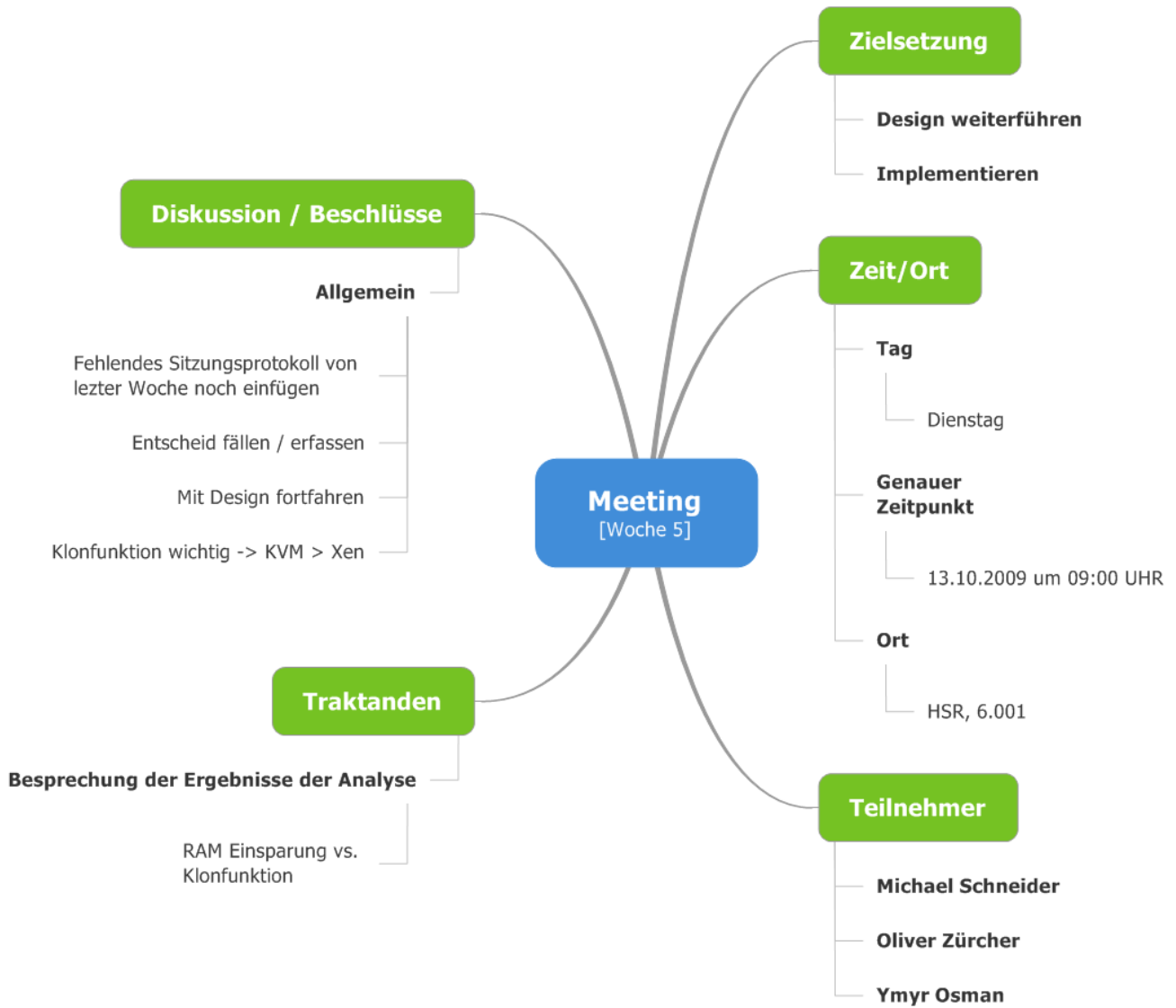
8.2 Woche 3



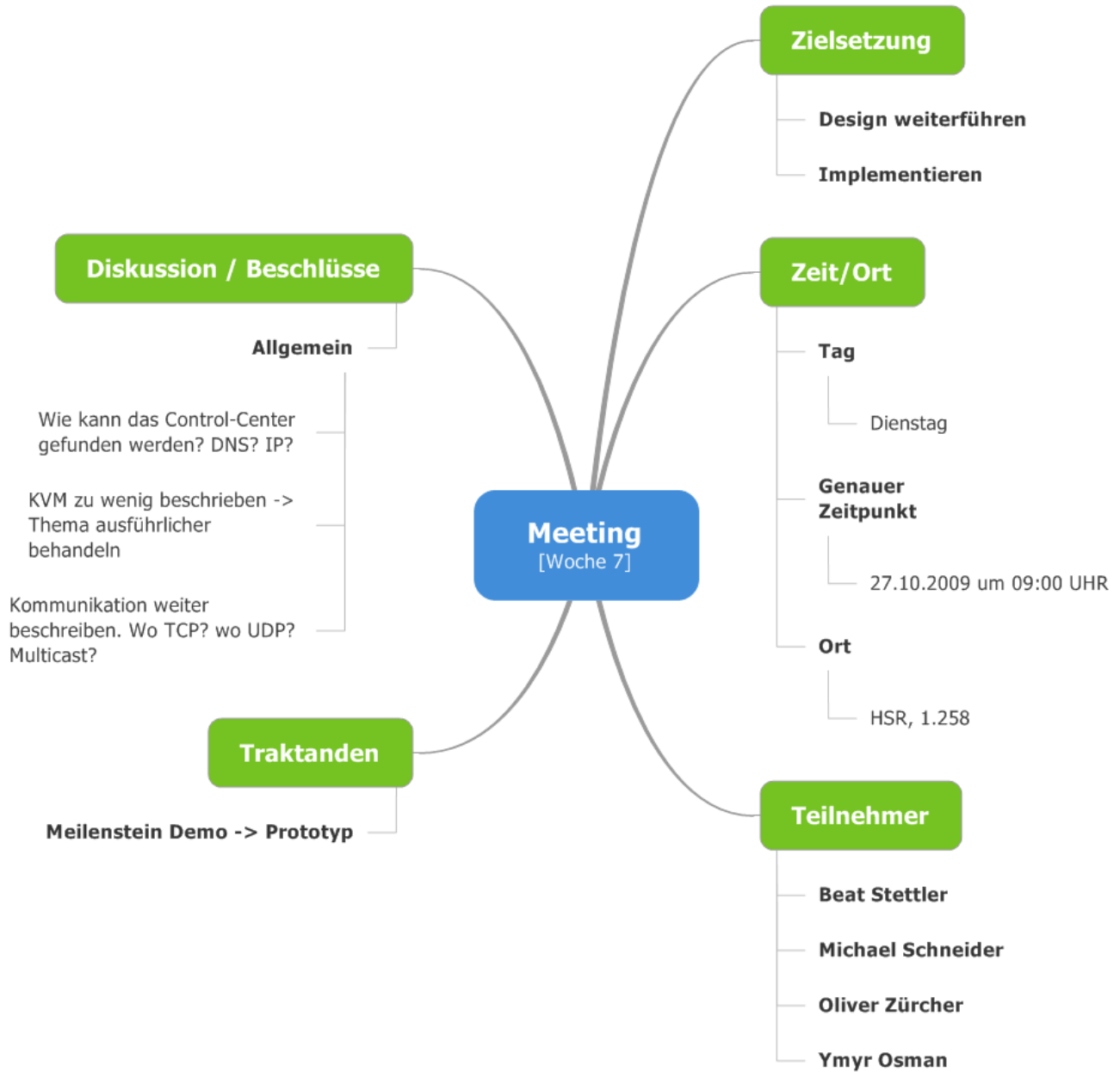
8.3 Woche 4



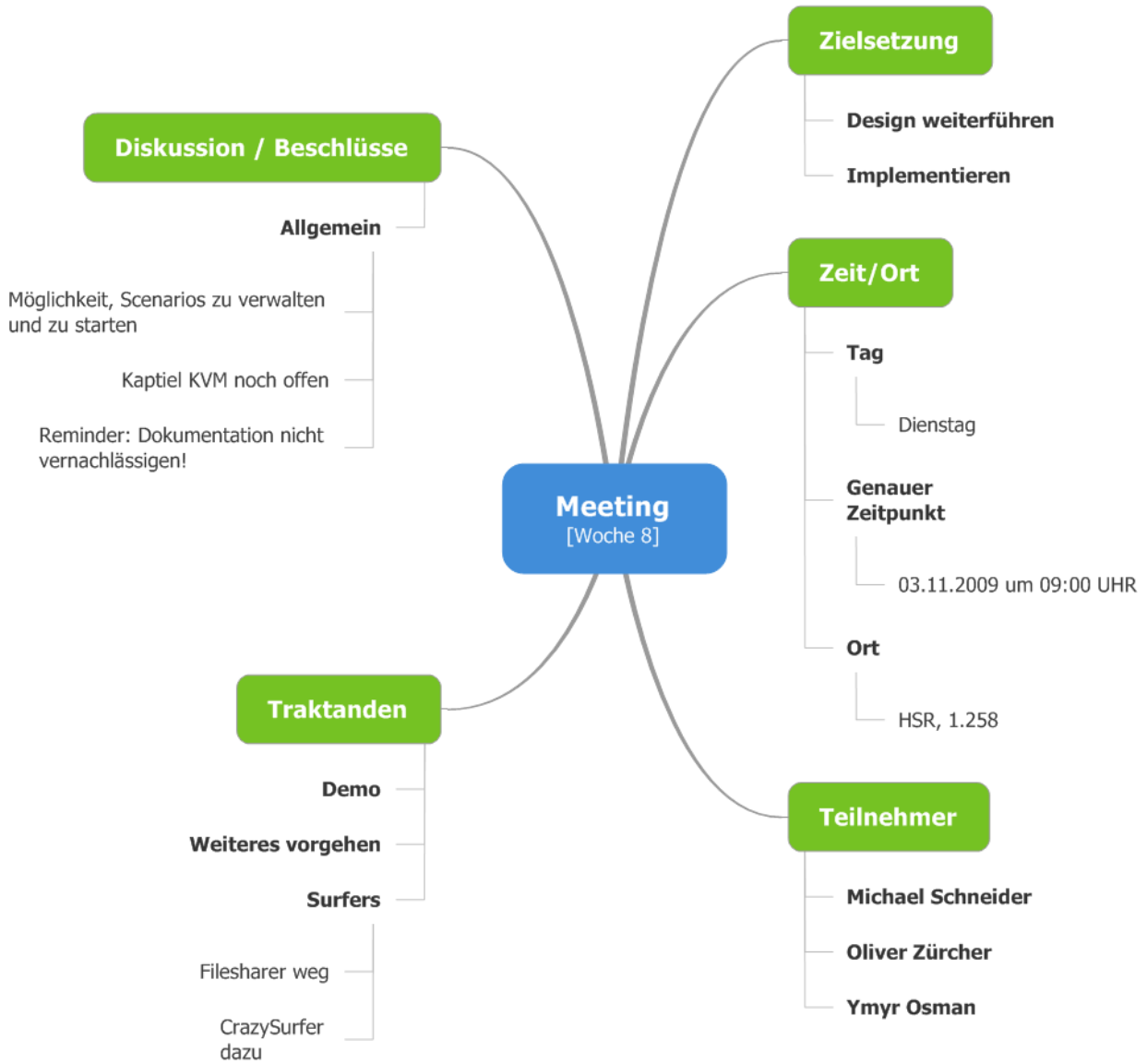
8.4 Woche 5



8.5 Woche 7 (MS Demo)



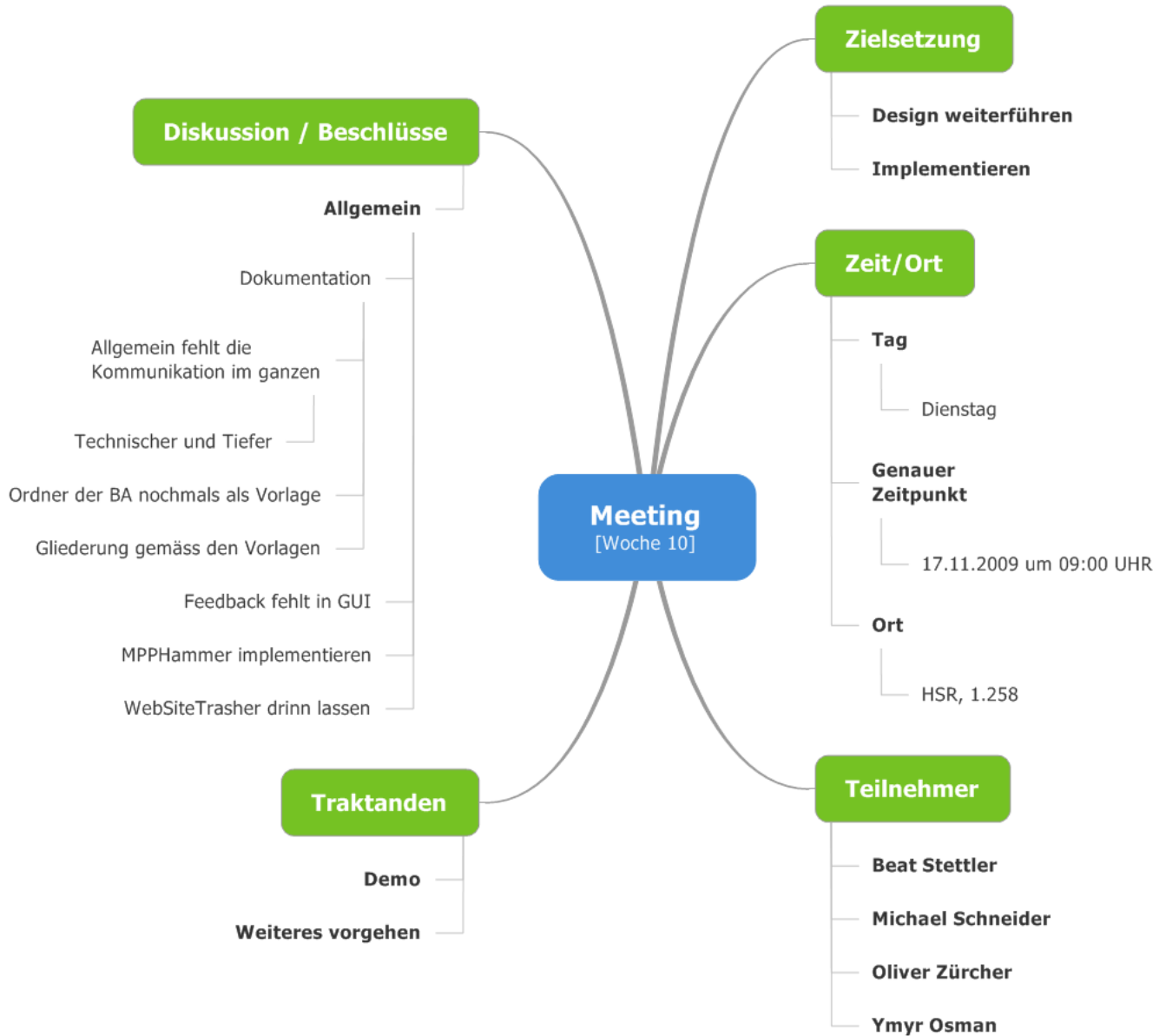
8.6 Woche 8



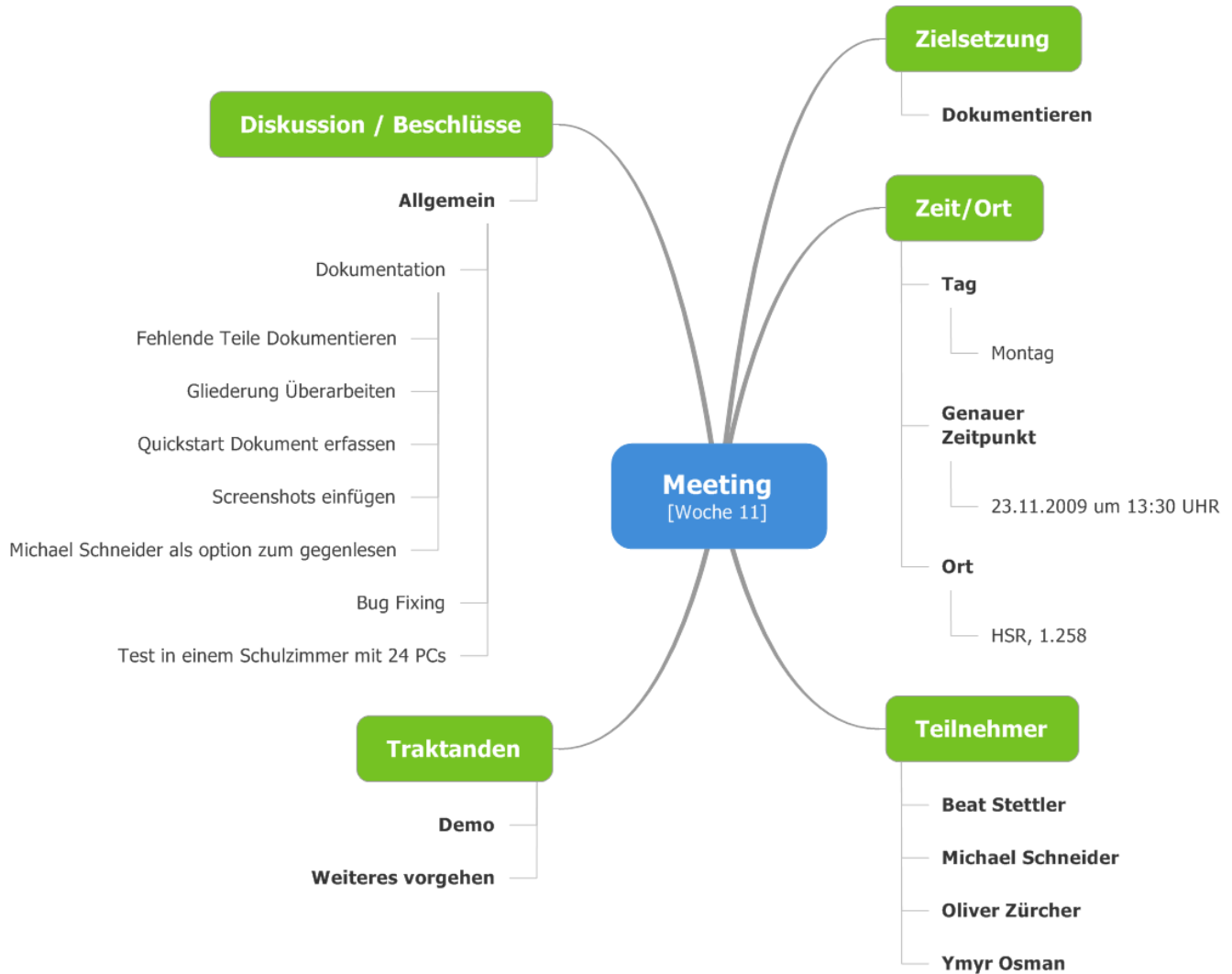
8.7 Woche 9



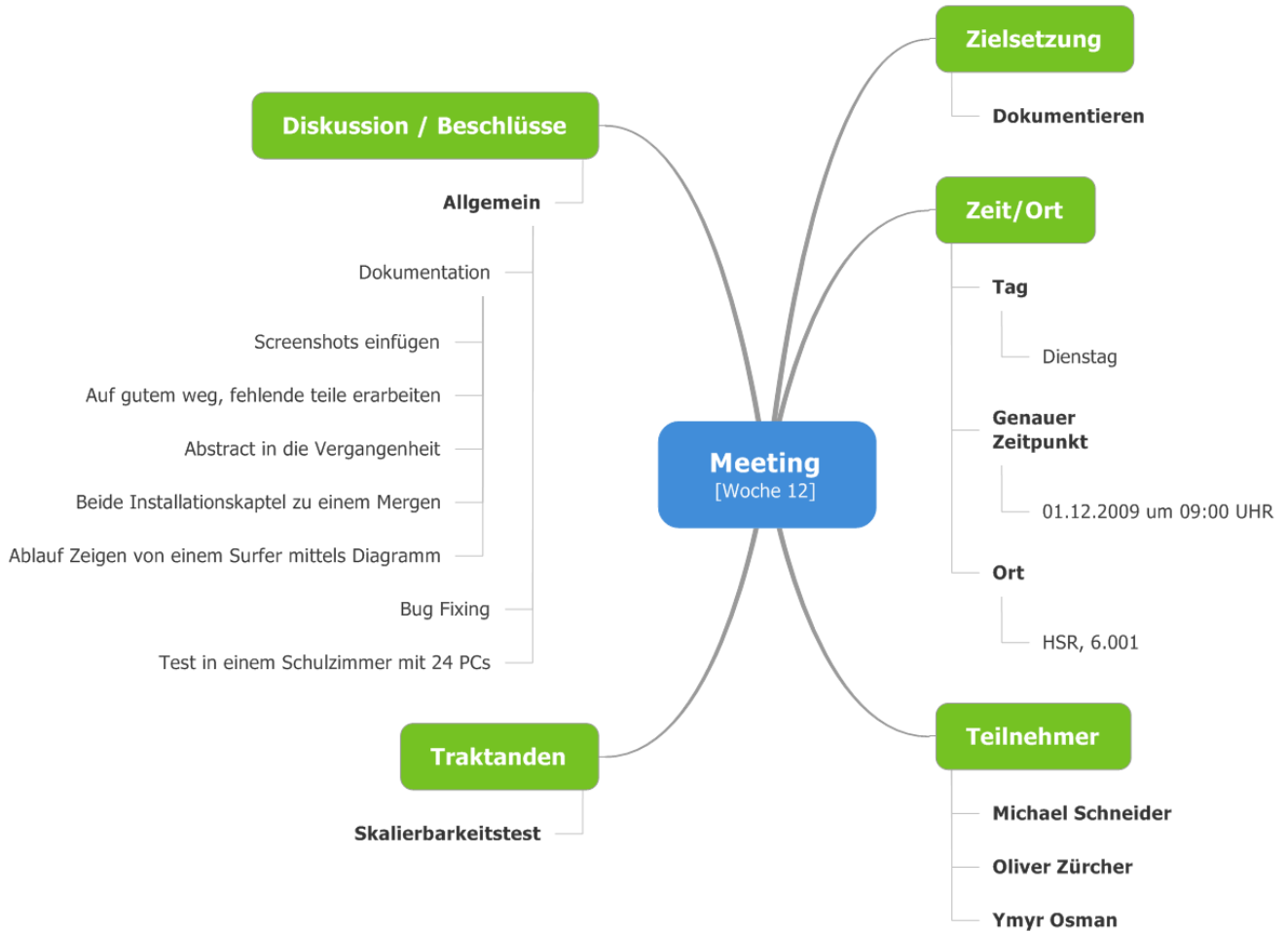
8.8 Woche 10



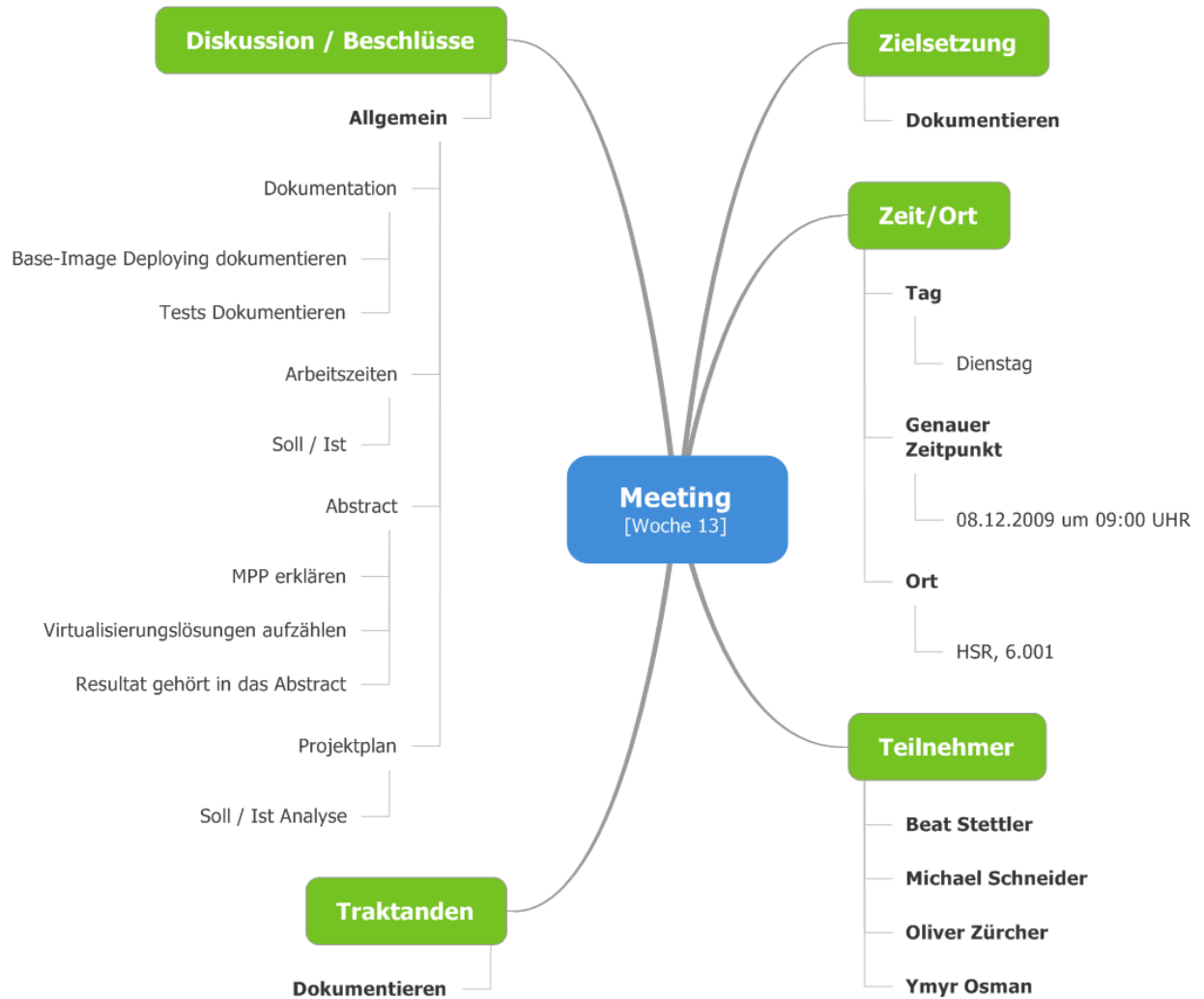
8.9 Woche 11



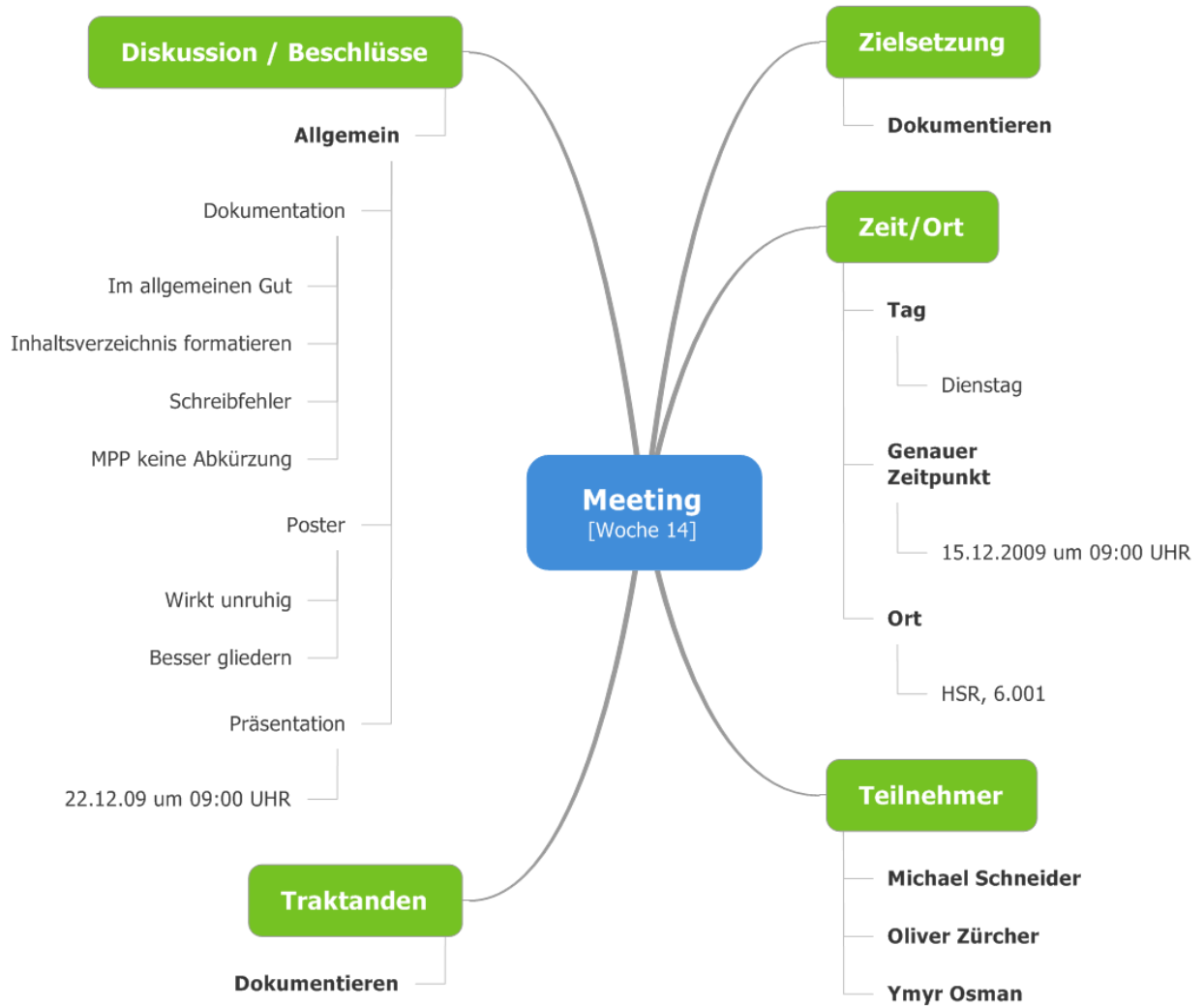
8.10 Woche 12



8.11 Woche 13



8.12 Woche 14



9 Abbildungsverzeichnis

Abbildung 1: Control-Center GUI	13
Abbildung 2: Zeitmessung sqrt()	17
Abbildung 3: Memory Test	17
Abbildung 4: Control-Center GUI	19
Abbildung 5: Remote Service Interface	20
Abbildung 6: Zeitplan	34
Abbildung 7: Zeitauswertung pro Iteration	43
Abbildung 8: Zeitauswertung pro Disziplin	44
Abbildung 9: Zeitauswertung pro Semesterwoche	44
Abbildung 10: Codeauswertung von Visual Studio	45
Abbildung 11: Use Cases	50
Abbildung 12: Zeitmessung sqrt()	60
Abbildung 13: Memory Test	60
Abbildung 14: 1. Visio Skizze von dem Control-Center GUI	65
Abbildung 15: 2. Visio Skizze von dem Control-Center GUI	66
Abbildung 16: 3. Visio Skizze von dem Control-Center GUI	67
Abbildung 17: Finale Skizze von dem Control-Center GUI	68
Abbildung 18: Beispiel einer Kommunikation eines Clients mit dem MPP	69
Abbildung 19: Architekturübersicht	70
Abbildung 20: Deployment Diagramm	72
Abbildung 21: Register Service Remote Interface	72
Abbildung 22: Beziehungen Control-Center - VMManager - ILoadGenerator	73
Abbildung 23: Status Remote Interface	73
Abbildung 24: Interface zur Steuerung der VMs auf der Node	73
Abbildung 25: Interaktion aller Komponenten	75
Abbildung 26: Logische Architektur	76
Abbildung 27: Scenario Klassendiagramm	78
Abbildung 28: XML Struktur der gespeicherten Szenarien	79
Abbildung 29: ILoadGenerator Schnittstelle - Wird von allen Surfers implementiert	85
Abbildung 30: Die wichtigsten Methoden der Web-Klasse	87
Abbildung 32: Klassendiagramm NewsReader	88
Abbildung 31: Klassendiagramm Googler	88
Abbildung 33: Klassendiagramm Downloader	89
Abbildung 34: Klassendiagramm MPPHammer	89
Abbildung 35: Control-Center GUI	106
Abbildung 36: Konsolenausgabe von VMRemoteControl.exe	107
Abbildung 37: Node Manager	107
Abbildung 38: Nodes triggern	108
Abbildung 39: Registrierte Nodes werden aufgelistet	108
Abbildung 40: Szenarioname	108
Abbildung 41: Einstellungen des Szenarios	109
Abbildung 42: Zeigt die Kapazität und Fortschritt an	109
Abbildung 43: Settings Dialog Downloader	110
Abbildung 44: Settings Dialog Googler	110
Abbildung 45: Settings Dialog NewsReader	110
Abbildung 46: Settings Dialog WebSiteTrasher	111

Abbildung 47: Control-Center Settings.....	111
Abbildung 48: Capacity Bar	112

10 Sourcecode Performance Messung

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Diagnostics;

namespace stress
{
    class Program
    {
        /// <summary>
        /// Zur Synchronisation für die Threads, welche auf die Konsole etwas ausgeben
        /// </summary>
        private static readonly object locker = new object();

        /// <summary>
        /// Einstiegsmethode
        /// </summary>
        /// <param name="args">Programmargumente</param>
        static void Main(string[] args)
        {
            int spinSqrt = 100000000;

            Thread t1 = new Thread(new ParameterizedThreadStart (SpinSqrt));
            t1.IsBackground = true;
            t1.Name = "SqrtSpinning Thread 1";
            Thread t2 = new Thread(new ParameterizedThreadStart (SpinSqrt));
            t2.IsBackground = true;
            t2.Name = "SqrtSpinning Thread 2";

            t1.Start ((object) spinSqrt);
        }
    }
}
```

```

        t2.Start((object) spinSqrt);

        t1.Join();
        t2.Join();

        SpinMemory(1024 * 1024 * 5);
        SpinMemory(1024 * 1024 * 20);

        Console.WriteLine("\nstress test finished.");
        Console.ReadLine();
    }

    /// <summary>
    /// Zieht die Wurzel von 1 bis n (n muss Int sein)
    /// </summary>
    /// <param name="o">Obere Grenze, muss in int gecasted werden können</param>
    private static void SpinSqrt(object o)
    {
        int n = (int)o;

        Stopwatch watch = new Stopwatch();
        Thread.Sleep(1);
        watch.Start();

        for(int i = 1; i < n; ++i)
            Math.Sqrt(n);

        watch.Stop();

        lock (locker)
        {
            Console.WriteLine(Thread.CurrentThread.Name + ": sqrt() 1.." + n + " " +
watch.ElapsedMilliseconds + "ms");
        }
    }

    /// <summary>
    /// Kopiert numberOfBytes Bytes 10mal in ein Buffer. Die Durchschnitts-

```



```

/// geschwindigkeit wird in MB/s auf der Konsole ausgegeben.
/// </summary>
/// <param name="numberOfBytes">Anzahl Bytes</param>
private static void SpinMemory(Int32 numberOfBytes)
{
    byte[] arr1 = new byte[numberOfBytes];
    byte[] arr2 = new byte[numberOfBytes];
    Stopwatch watch = new Stopwatch();

    List<double> results = new List<double>();

    for (int i = 0; i < numberOfBytes; ++i)
        arr1[i] = 0;

    for (int i = 0; i < 10; ++i)
    {
        watch.Reset();
        Thread.Sleep(1);
        watch.Start();
        arr1.CopyTo(arr2, 0);
        watch.Stop();

        double speed = (double)((double)numberOfBytes / 1024.0) /
(double) (watch.ElapsedMilliseconds);
        results.Add(speed);
    }

    Console.WriteLine(string.Format("avg speed writing {0:F2} MB blocks: {1:F2} MB/s",
        (double)numberOfBytes / 1024.0 / 1024.0 ,
        results.Average()));
}
}

```