

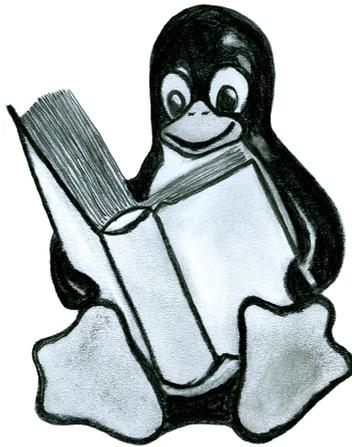
**Team der Chemnitzer Linux-Tage:**  
**Chemnitzer Linux-Tage 2012**  
– Tagungsband –  
17. und 18. März 2012



Team der Chemnitzer Linux-Tage

# Chemnitzer Linux-Tage 2012

17. und 18. März 2012



– Tagungsband –



TECHNISCHE UNIVERSITÄT  
CHEMNITZ

Universitätsverlag Chemnitz  
2012

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über <http://dnb.d-nb.de> abrufbar.

Technische Universität Chemnitz/Universitätsbibliothek  
Universitätsverlag Chemnitz

Herstellung und Auslieferung:  
Verlagshaus Monsenstein und Vannerdat OHG  
Am Hawerkamp 31  
48155 Münster  
<http://www.mv-verlag.de>

ISBN 978-3-941003-52-1

URL: <http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-83272>

Satz und Layout: Jens Pönisch und Monique Kosler  
Covergraphik: Candida Winkler  
Graphik Innentitel: Petra Pönisch  
URL der Chemnitzer Linux-Tage: <http://chemnitzer.linux-tage.de>

## Premiumsponsoren



Linux Lösungen Lübeck



## Weitere Sponsoren



## Medienpartner





# Inhaltsverzeichnis

<b>1</b>	<b>Vorwort</b>	<b>11</b>
<b>2</b>	<b>Inhalt der Hauptvorträge</b>	<b>13</b>
2.1	AnyOS	13
2.2	Bitcoin – das elektronische Geld	21
2.3	Busverkehr unter Linux	35
2.4	Entwicklung cyber-physikalischer Systeme: der NAO-Roboter	45
2.5	Gründen mit freier Software	53
2.6	Lilypond – ein wenig Revolution muss sein	61
2.7	MapReduce – Parallelität im Großen und im Kleinen	69
2.8	Performancemessungen und -optimierungen an GNU/Linux-Systemen	77
2.9	Starke Zweifaktorauthentisierung mit LinOTP	87
2.10	Strace für Bash-Versteher	99
2.11	Tic-Tac-Toe Reloaded – Mikrocontrollerprojekt mit Funkübertragung	109
2.12	Ubiquitos Computing	117
2.13	Visual Scripting: Scripting the Unscriptable	131
2.14	Zero Commercial Software Strategy – eine Fallstudie	137
<b>3</b>	<b>Zusammenfassungen der weiteren Vorträge</b>	<b>147</b>
3.1	Adaptive tickless kernel	147
3.2	Aktuelle Entwicklungen beim Linux-Kernel	147
3.3	APT – Paketverwaltung für Debian GNU-Linux	147
3.4	Back to Unix: 100 POSIX-Kommandos statt 29.000 Packages	148
3.5	Bash the Publisher – Vom Sinn und Unsinn des Verlagswesens	148
3.6	Best Practice OTRS – IT Service Management ganz praktisch	148
3.7	Conkeror und andere tastaturfokussierte Webbrowser	148
3.8	Continuous Integration mit Jenkins für Perl-Projekte	149
3.9	CouchDB und CouchApps	149
3.10	crypto.is	149
3.11	Das Anlagen- und Indirekteinleiterkataster der Stadt Bielefeld	149
3.12	Datenbanken von MySQL zu PostgreSQL portieren	150
3.13	Datensicherheit durch Datensicherung mit bacula	150
3.14	Der einfache Umstieg auf Linux mit Kubuntu	150
3.15	Der Systemaufruf und was danach passiert	150
3.16	Developing Linux inside QEMU/KVM Virtual Machines	151
3.17	Die ArchivistaBox kurz erklärt	151
3.18	Die Entwicklung von Linux durch Testen neuer Kernel unterstützen	151
3.19	Die Helfer der Kommandozeile	151
3.20	Die Macht der Zahlen – Open-Source-Tools für Softwaremetriken	152

3.21	E-Books mit offenen Standards realisieren	152
3.22	Einfaches Bauen von RPM-Paketen	152
3.23	Einführung in die 3D-Visualisierung mit Blender	152
3.24	Fedora Security Lab and the OSSTMM	153
3.25	Free your slides – Vortragsfolien im Browser anzeigen	153
3.26	Freie Software und Offene Standards – ein starkes Team	153
3.27	Freifunk – drahtlose Community-Netzwerke	153
3.28	FTrace and KernelShark	154
3.29	Gib SPAM keine Chance – wirkungsvolle Spamabwehr mit postscreen	154
3.30	Globale Wissenstransferprojekte mit KnowledgeWorker	154
3.31	Grep everything – geschicktes Suchen in Anwendungsdaten	154
3.32	Hochverfügbarkeit und Virtualisierung – die optimale Kombination beider Konzepte	155
3.33	Integrationstests am Beispiel des Linux-Kernels	155
3.34	Konfiguration und Analyse von Kernel Crashdumps	155
3.35	Konfigurations- und Infrastrukturmanagement mit RPM und YADT	156
3.36	Linux für die kaufmännischen Arbeiten im Unternehmen	156
3.37	LinuxContainer in der Praxis	156
3.38	Live-Demo: UCS 3.0 und Samba-4-Integration	156
3.39	LXC oder Jails und Zones	157
3.40	Mailserver und Spamschutz bei IPv6	157
3.41	Menschen reden anders als Maschinen – gepflegter Umgang mit Kritik	157
3.42	Mit OpenStack zur eigenen Cloud	157
3.43	Mixare.org – eine Augmented Reality Engine	158
3.44	Monitoring mit Zabbix	158
3.45	Neue Farbtrends	158
3.46	NGoC mit Zarafa	158
3.47	Non-free software advertisement presented by your government	159
3.48	Open Data Community für öffentliche Verwaltungen und Unternehmen	159
3.49	OpenAFS – ein Überblick für Neulinge	159
3.50	OpenAFS aus Anwender- bzw. Klientensicht	159
3.51	OpenAFS-Administration – die andere Seite	160
3.52	OpenAFS-Authentisierung – das Kerberos-Basisprotokoll zum Anfassen	160
3.53	Penetration Testing mit Metasploit	160
3.54	Programmierung für Smartphones – die Lizenzen für Android und iOS erläutert	160
3.55	Python – Open-Source-Werkzeug für Wissenschaftler und Ingenieure	161
3.56	QEMU's recompilation engine	161
3.57	Qualitätsanalyse und Teammanagement in Open-Source-Projekten	161
3.58	Rechnungseingangsverarbeitung mit Alfresco	161
3.59	SAN Storage on Linux	162
3.60	SocketCAN – CAN-Treiberschnittstelle unter Linux	162
3.61	Spacewalk	162
3.62	Speichereffiziente Datensicherung mit Dirvish	162
3.63	Stop-Motion-Trickfilme mit Linux	163

3.64	Supercomputing and Open Source after the K-Computer . . . . .	163
3.65	TaskJuggler 3.0 – Projektmanagement für Linux-Anwender . . . . .	163
3.66	The 3.0-rt Kernel . . . . .	163
3.67	Tine 2.0 – Open-Source-Groupware und CRM . . . . .	163
3.68	Typo3 für Entwickler . . . . .	164
3.69	Ubuntu im sicheren privaten, virtuellen Netz . . . . .	164
3.70	Ursprünge der Versionsverwaltung und Revival von SCCS . . . . .	164
3.71	Virtual System Cluster: Freie Wahl mit Open Source . . . . .	165
3.72	Virtualisierung @ Google . . . . .	165
3.73	Vom Piep zum Boot – BIOS und Co. . . . .	165
3.74	Web Framework Django . . . . .	165
3.75	Wollmux trifft auf SAP & Co – dynamische Reports mit dem freien Vorlagensystem . . . . .	166
3.76	Zentrales Konfigurationsmanagement mit Puppet . . . . .	166
<b>4</b>	<b>Zusammenfassungen der weiteren Workshops</b>	<b>167</b>
4.1	Arduino – Ausbruch aus dem Compile-Upload-Debug-Kreislauf . . . . .	167
4.2	Einführung in Blender . . . . .	167
4.3	Einstieg in die Betriebssystementwicklung – Grundlagen für das eigene OS . . . . .	167
4.4	FreeBSD-Installation und -Konfiguration . . . . .	168
4.5	Inkscape – Eiszeit . . . . .	168
4.6	KMUX – Installation der IT-Landschaft für ein KMU in 2 Stunden . . . . .	168
4.7	Offenes Storage Management mit openATTIC . . . . .	168
4.8	OpenAFS – eine eigene AFS-Zelle aufsetzen . . . . .	168
4.9	OpenFOAM: Numerik mit freier Software am Beispiel . . . . .	169
4.10	OpenStreetMap . . . . .	169
4.11	Python für Einsteiger . . . . .	169
4.12	Seeing inside the Linux Kernel with ftrace . . . . .	169
4.13	Starbasic – eine Einführung . . . . .	170
4.14	Vim-Führerschein: Grundlegendes Editieren mit Vim . . . . .	170
<b>5</b>	<b>Personen</b>	<b>171</b>



# 1 Vorwort

Stabilität und Zuverlässigkeit von Linux-Systemen werden schon lange nicht mehr bezweifelt. Eher steht noch die Stabilität der Unterstützung in Frage. Welche Projekte werden auch in den kommenden Jahren noch kraftvoll weiterentwickelt? Auf welche Entwicklungen kann man sich in langfristigen Planungen verlassen? Wird der Kernel auch im nächsten Jahr noch die aktuelle Hardware unterstützen?

Das spiegelt sich auch in den Themen der Linux-Tage wider. Viele eingereichte Beiträge suchen nach Möglichkeiten, ein Projekt am Leben zu erhalten und Mitstreiter zu finden. Das sind Fragen, denen sich auch die Veranstaltung selbst immer wieder stellen muss. Wenn Sie diesen Band in den Händen halten, ist uns das ein weiteres Mal gelungen. Dafür gebührt einer Vielzahl von Mitwirkenden großer Dank: Helfer, Organisatoren, Referenten, Aussteller haben oft in mehreren Funktionen an der Vorbereitung und Durchführung mitgewirkt.

Das Thema Freie Software – nicht nur mit Blick auf Linux – brennt noch immer. Der ungehinderte Zugang zu den Quellen macht sie als Werkzeug für viele weitere Aufgaben interessant. Die Hürde um neue Horizonte zu erschließen ist dadurch gering: Ob man beispielsweise standardisierte Bussysteme kennenlernen oder einen Mikrocontroller programmieren will – mit wenigen Euro Investition steht die Versuchsanordnung. Das fasziniert, und das sorgt für Nachwuchs.

Die Chemnitzer Linux-Tage sollen Horizonte erweitern und damit auch Wissen und Methoden anderer Projekte verfügbar machen. So entstehen Synergien, die wir dringend benötigen, denn Entwickler sind immer knapp – siehe oben. Schön wäre es, wenn dieser Band diese Anregungen geben kann – in den ausführlichen Artikeln oder in den Vorträgen.

Der ausführliche Kernel-Track zeigt, dass die Chance zum Austausch angenommen wird. Das schafft Vertrauen. Die Kraft des Linux-Kernels strahlt aus, viele große Projekte wären ohne diese Basis nie gewachsen, selbst wenn sie jetzt auf mehreren Plattformen zu Hause sind. Kernelkraft und erneuerbare Synergien – das ist nicht nur ein Motto, das sind die Chemnitzer Linux-Tage.

Ralph Sonntag im Februar 2012



# AnyOS

Axel Schöner, Wilhelm Meier

axel.schoener@fh-kl.de

<http://www.fh-kl.de/~axel.schoener>

In diesem Artikel über AnyOS [1] wird ein Konzept beschrieben, mit dessen Hilfe sich der Aufwand bzgl. der Verwaltung und Wartung von Rechnersystemen jeglicher Betriebssysteme mittels gängiger Techniken minimieren lässt.

Die gleichzeitige Verwendung mehrerer grafischer Betriebssysteme auf einem Rechner erhöht die Flexibilität für den Einsatz unterschiedlicher Software.

Die Idee zu AnyOS stammt von Wilhelm Meier um wichtige Anforderungen zur Integration von KMUX [2] in bestehende Unternehmen zu erfüllen. KMUX ist ein Open-Source-Konzept für kleine und mittlere Unternehmen um eine einfache und umfassende EDV-Infrastruktur bereitzustellen. Durch die Integration von Anwendungen zur Erleichterung des Unternehmensalltags, bietet es eine solide Basis zur Erweiterung und Anpassung auf die eigenen Bedürfnisse.

## 1 Einführung

Die Wartung und Pflege einer großen Anzahl von PC-Installationen ist in der Regel eine Aufgabe, die großen Aufwand nach sich zieht. Eine Möglichkeit, diesen Administrationsaufwand zu reduzieren, ist die Einführung von Thin-Clients [3] und entsprechender Serverkapazitäten.

Gerade aber in Migrationsszenarien ist damit ein erheblicher Investitionsbetrag verbunden. Gleichzeitig geht dabei die Flexibilität der PC-Nutzung verloren. Beide Probleme können jedoch durch eine intelligenter Nutzung der vorhandenen Ressourcen minimiert werden.

Zum Einen können vorhandene PC-Systeme mit geringem Aufwand zu Linux-Thin-Clients umkonfiguriert werden. Dazu existieren seit langem vorkonfigurierte Linux-Distributionen. In diesem Fall werden aber nach wie vor entsprechende Serverkapazitäten benötigt, um die angestrebte Anzahl von gleichzeitigen Sitzungen auf dem/den Terminalserver(n) zu ermöglichen.

Andererseits wesentlich effizienter ist die Verwendung von sogenannten Rich-Clients [4]. Hier laufen die Applikationen lokal auf dem Client ab, allerdings ohne eine Installation auf dem Client vorauszusetzen.

Im Rahmen von AnyOS wird eine flexiblere, zustandslose, netzwerk-boot Rich-Client Lösung vorgestellt. Damit wird die Wartung zentralisiert und extrem vereinfacht und gleichzeitig Investitionen geschützt. Selbstverständlich ist auch ein Mischbetrieb Thin-/Rich-Client machbar.

Um nun auch die Welt der non-Linux Betriebssysteme auf diesen Rich-Clients lauffähig zu machen, kommt ein reduzierter Rich-Client zum Einsatz, der als Basis für KVM [5]/qemu [6] dient. Die notwendigen (virtuellen) Festplatten werden vom Server als Copy-On-Write-Abbilder [7] jeder KVM-Instanz netzwerkbasierend bereitgestellt. Die

graphische Ausgabe des emulierten Systems wird über SDL/VNC/RDP vom Rich-Client lokal dargestellt, welche im weiteren Verlauf als Viewer bezeichnet werden. Auf diese Weise ist es möglich, jedes Betriebssystem, das in KVM/qemu lauffähig ist, in einer beliebigen Anzahl von Instanzen ablaufen zu lassen. Durch den zustandslosen Ansatz sind diese Instanzen "unkaputtbar", eine weitere Pflege ( zum Beispiel Datenträgerbereinigung und Virenprüfung) entfällt! Für das Gastsystem ist der Ansatz vollständig transparent. Damit können auch Systeme, die originär nicht netzwerkbootfähig sind (beispielsweise Microsoft Windows), auf den Rich-Clients ablaufen. Bei moderner PC-Hardware mit vollständiger Virtualisierungsunterstützung (Intel VT or AMD-V) und aktuellen Netzwerkkomponenten ist eine Beeinträchtigung der Ablaufgeschwindigkeit nicht feststellbar. Über einen Netzwerk-BootManager kann eine Auswahl des Gastsystems getroffen werden, wobei jede nach Leistungsfähigkeit auch mehrere Gastsysteme auf einem physischen Rich-Client gestartet werden können. Zudem bleibt selbstverständlich die Möglichkeit, neben den virtualisierten Gastsystemen den Linux Rich-Client mit Applikationen auszustatten.

Das AnyOS-System ist ideal für beispielsweise Schulungs- und Testumgebungen. Hier müssen typischerweise eine große Anzahl identischer Systeme gleichzeitig bereitgestellt werden. Darüberhinaus besteht zumeist die Anforderung sehr viele unterschiedliche Systeme zur Verfügung zu haben. Beides ist mit AnyOS einfach und effizient realisierbar. In diesem Szenario ist AnyOS an der Fachhochschule Kaiserslautern / Zweibrücken im Einsatz.

Ein anderer Einsatzbereich ist sehr oft in kleinen und mittelständischen Unternehmen anzutreffen, denn hier ist zwar oft eine OpenSource-Migration gewünscht, allerdings sind branchenspezifische Applikationen vielfach in Linux-Umgebungen nicht lauffähig, wie zum Beispiel Microsoft Outlook. Im Zusammenhang mit KMUX [2] wurde dieses Szenario bei sehr vielen Unternehmen festgestellt woraus die Idee zu AnyOS entstand. Weiterhin lassen sich bekannte Inkompatibilitäten bestimmter Programme oder Versionen umgehen. In diesem Fall kann die Virtualisierung der Altsysteme effizient mit AnyOS erfolgen.

## 2 Gewünschte Eigenschaften eines solchen Systems

- Jegliches Betriebssystem soll nach dem gleichen Prinzip nutzbar sein.
- Erstellung einer Basisinstallation, welche sich um weitere Software erweitern lässt.
- Die Installationen sollen auf allen verfügbaren Rechnern ausgeführt werden können.
- Unterschiedliche Anforderungen seitens der Anwender sollen berücksichtigt werden können.
- Jeder Rechner soll beim Neustart ein frisches, sauberes System starten.

- Wartungsarbeiten sollen parallel zum Nutzbetrieb ausgeführt werden können.
- Modifizierte Konfigurationen sollen vor Inbetriebnahme ausführlich getestet werden können.
- Veränderungen am System sollen jederzeit rückgängig machbar sein.
- Die unterschiedlichen Systeme sollen möglichst speichereffizient vorgehalten werden.
- Die Ansprüche an erforderliche Hardware (Client und Server) sollen möglichst gering sein.

### 3 Technische Umsetzung

Das Setup besteht aus einer Server- sowie einer Clientkonfiguration. Folgende Dienste werden auf dem Server betrieben:

- DHCP-Server, um das Netzwerk für die Clients zu konfigurieren und um Ihnen die Adresse des TFTP-Servers mitzuteilen.
- TFTP-Server, um die Clients über Netzwerk booten zu können. Dieser bietet anfragenden Clientrechnern einen Bootmanager "pxelinux" sowie auf Anfrage den Kernel und falls notwendig eine Ramdisk an.
- NFS-Server, zum Exportieren des Betriebssystems für den Bootvorgang der Clientrechner, basierend auf Ubuntu.
- NBD-Server, zum Exportieren der zu virtualisierenden Betriebssystemimages.

Die Clientkonfigurationen werden vom Server für die vorgesehenen Rechner im Netzwerk verfügbar gemacht. Als Grundlage zum Betrieb der gewünschten Betriebssysteme dient ein angepasstes, minimales Ubuntu, das klassisch über Netzwerk als Rich-Client gestartet wird (TFTP + NFS) und alle clientseitigen Aktionen mittels Upstart-Skripten [8] triggert.

Um sowohl von der Basisinstallation des Rich-Clients als auch der zu betreibenden Betriebssysteminstallationen angepasste Installationen effizient ableiten zu können, wird BTRFS als Dateisystem verwendet. Das Fehlen eines geeigneten Filesystem-checks für BTRFS ist für das Einsatzszenario kein Hindernis, da alle Konfigurationen rein lesend verwendet werden. Lediglich temporäre Snapshots, die innerhalb einer Clientsitzung erstellt werden, erfordern Schreibzugriff. Um verschiedene Konfigurationen je nach Bedarf anbieten zu können, empfiehlt sich die Erstellung von Snapshots des bereits erwähnten Rich-Clients, in dem die gewünschten Änderungen vorgenommen werden. Alternativ lässt sich dies mittels Overlay-Dateiesystemen

wie zum Beispiel aufs [9] realisieren. Für jede Variante ist eine TFTP- sowie NFS-Konfiguration zu erstellen. Dies ermöglicht abhängig vom Rechner beziehungsweise der Auswahl bestimmte Konfigurationen zu starten.

Die zu virtualisierenden Installationen selbst werden über NBD als Bootlaufwerke exportiert. Für jeden Client wird beim Einbinden der NBD-Freigabe mittels COW-Mechanismus ein temporärer Snapshot von der gewünschten Installation abgeleitet, um Schreibzugriffe zuzulassen. Dieser Vorgang ist besonders effizient, da BTRFS hierzu lediglich geänderte Daten schreibt. Zusätzlich zu den Bootlaufwerken können Swaplaufwerke exportiert werden, um fehlenden Arbeitsspeicher auf Clientseite auszugleichen.

Zum Vorhalten von Daten für den Benutzer sowie dessen Authentifizierung empfehlen sich gängige Netzwerklösungen, wie zum Beispiel `openldap` [10] und `Samba` [11]. Die Installationen werden nicht nativ auf der Hardware betrieben, sondern innerhalb einer über `KVM/qemu` emulierten Hardware. Dies ermöglicht die Verwendung jedes Betriebssystems und schafft eine einheitlichen Betriebsumgebung. Zum Zugriff auf diese emulierten Systeme wird für jede Betriebsumgebung ein `X` Server an ein virtuelles Terminal gebunden, über den mittels `Viewer` auf die grafischen Systeme zugegriffen wird. Alle Tastatur- und Maus-Eingaben werden dabei innerhalb des virtuellen Systems umgesetzt. Audioausgaben lassen sich mittels virtueller Soundkarte über den Host ausgegeben.

Unter Verwendung mehrerer Instanzen von `KVM/qemu`, `X` Server und `Viewer` lassen sich mehrere und/oder unterschiedliche Betriebssysteme gleichzeitig auf verschiedenen virtuellen Terminals betreiben.

## 4 Einrichtung des Grundsystems

Auf dem zu verwendenden Server wird eine minimale Installation von Ubuntu-Server vorausgesetzt. Unter [1] sind die Installationsdateien von AnyOS herunterzuladen. Durch Aufruf des Skriptes `install_server.sh amd64/i386` wird die komplette Konfiguration des Servers sowie eines ersten Rich-Clients vorgenommen. Weitere Rich-Clients können wie folgt angelegt werden:

```
# ./snapshot.sh -r client_amd64.base01 $snapshotname
# ./prepare.sh $snapshotname
# ./populate.sh -r $snapshotname
```

Durch Verwendung dieser Skripte wird ein Snapshot vom bereits existierenden Rich-Client 'client\_amd64.base01' erstellt sowie die Konfiguration von (s)chroot und NFS vorgenommen. Anzupassen ist lediglich die TFTP-Konfiguration sowie die Konfigurationsdateien zu den gewünschten virtuellen Betriebsumgebungen (s.a. 6).

## 5 Betriebssystemimage erstellen

Um ein virtuelles Betriebssystem zu installieren legt man zuerst ein Subvolume für dieses an:

```
# btrfs subvolume create /tftpboot/qemu/ubuntu/
```

Nun erzeugt man ein Festplatten-Image:

```
# qemu-img create /tftpboot/qemu/ubuntu/ubuntu.img 5G
```

alternativ mit:

```
# dd if=/dev/zero of=/tftpboot/qemu/ubuntu/ubuntu.img count  
=5120 bs=1024k
```

Die Installation der zu virtualisierenden Betriebssysteme kann man per VNC oder SDL vornehmen, dazu bedient man sich einem der folgenden Kommandos:

```
# qemu -hda ubuntu.img -cdrom ubuntu-11.04-desktop-i386.iso  
-smp 2 \  
-k de -boot d -m 1024 -vnc :1
```

```
# qemu -hda ubuntu.img -cdrom ubuntu-11.04-desktop-i386.iso  
-smp 2 \  
-k de -boot d -m 1024
```

Weitere mögliche Parameter können über "\$ qemu -help" oder "\$ man qemu" eingesehen werden.

Wird VNC verwendet, installiert man über einen vncviewer lokal das grafische System:

```
$ vncviewer 127.0.0.1:1
```

Unter Verwendung von SDL startet sich automatisch eine grafische Darstellung, sofern bereits ein X Server läuft.

Zu prüfen ist, ob das installierte Betriebssystem einen Treiber für den Netzwerkkartenadapter Intel e1000 benötigt, da dieser für die virtuelle Netzwerkkarte benötigt wird. Dieser sollte falls erforderlich installiert werden. Bei Windows XP liegt zum Beispiel der erforderliche Treiber nicht vor, dieser kann von der Intel-Webseite heruntergeladen werden [12]. Für die Installation von Ubuntu ist dies nicht erforderlich.

Um RDP bei Windows-Betriebssystemen verwenden zu können, ist dieses über die Systemeinstellungen zu aktivieren.

Soll das so erzeugte Image mehrfach auf einem Host laufen bzw. in veränderter Form, so erstellt man dafür am Besten einen neuen Snapshot.

## 6 Konfiguration der Clients

Nachdem das Clientimage bereit ist, passt man die NBD-Konfiguration an, damit dieses auch per Netzwerk exportiert wird. Dazu bearbeitet man für den Export die Datei `/etc/nbd-server/config` und für das Einbinden des Laufwerks die Datei `nbd-client` unter `/tftpboot/client64.rich01/etc/`.

Zu jeder zu startenden virtuellen Instanz bedarf es noch der zugehörigen Konfigurationsdateien. Diese befinden sich unter: `/tftpboot/client64.rich01/etc/init/`. Für die erste Instanz handelt es sich dabei um die Dateien `qemuinstance01.conf`, `xserver01.conf` sowie `viewer01.conf`. Zum Verständnis des groben Aufbaus der Datei `qemuinstance01.conf` hier ein Auszug mit den konfigurierbaren Parametern:

```
# /etc/init/qemuinstance01.conf
description    "Start a qemu instance"
start on (started networking and started qemu-kvm and
          started xserver01)
stop on runlevel [!2345]
respawn

env num=0      # DISPLAY={num}.0
env device=/dev/nbd1
env mem=320M
env nicmodel=e1000
env forwardip=127.0.0.1
env options="-k de -usbdevice tablet -enable-kvm -full-
            screen -no-acpi -soundhw ac97"
env graphic=vnc      # sdl | vnc | qxl | rdp
...
post-start script
logger -t "qemuinstance" "post start sleep"
sleep 1
end script
```

Weitere Instanzen konfiguriert man, indem man einen Symlink (symbolischer Link) auf diese Dateien erstellt mittels:

```
# ln -s qemuinstance01.conf qemuinstance02.conf
# ln -s xserver01.conf xserver02.conf
# ln -s viewer01.conf viewer02.conf
```

Zum Überschreiben der abweichenden Parameter, erstellt man entsprechende Dateien mit gleichem Namen, welche jedoch nicht auf `.conf` sondern auf `.override` enden. Die notwendigen Parameter befinden sich jeweils am Anfang der betreffenden Dateien.

Der Parameter `num` spielt dabei eine besondere Rolle. Diesen gilt es mit jeder weiteren Instanz zu inkrementieren. Von diesem Parameter hängen etliche andere Parameter ab. Er beeinflusst die Umgebungsvariable `$Display` sowie die Ports, die zum Betrieb der virtuellen Umgebung notwendig sind. Zur leichteren Konfigurierbarkeit wurden für die möglichen Zugriffsprotokolle sämtliche notwendigen Einstellungen vorbereitet, so dass nur der Parameter "graphic" verändert werden muss. Diese beiden Parameter sind in jeder der drei zugehörigen Konfigurationsdateien anzupassen: `gemuinstance*.conf`, `viewer*.conf` sowie `xserver*.conf`.

Um den Clients beim Starten die entsprechenden Konfigurationen anzubieten, sind diese in der Datei `boot/pxelinux.cfg/default` zu konfigurieren basierend auf der Pfadangabe, die dem TFTP-Server zugewiesen ist. Sollen aufgrund der MAC-Adressen bestimmter Rechner gesonderte Konfigurationen angeboten werden, so können für diese eigene Konfigurationsdateien innerhalb `boot/pxelinux.cfg/` erstellt werden.

## 7 Bedienung

Nach erfolgreichem Bootvorgang gelangt der Benutzer automatisch zu einem grafischen Betriebssystem. Wenn mehrere Betriebssysteme konfiguriert sind, kann der Benutzer zwischen diesen über Auswahl des jeweiligen virtuellen Terminals, mit der Tastenkombination: "strg + alt + Fn" umschalten. Auf diese Weise kann zwischen maximal 24 unterschiedlichen Gastbetriebssystemen umgeschaltet werden (für VT13-VT24 ist statt der linken 'strg'-Taste die rechte 'strg'-Taste zu verwenden), sofern auf den Zugriff für den physischen Client per virtuellem Terminal verzichtet wird.

## 8 Mögliche Vorgehensweise für Wartungsarbeiten

Durch folgende Vorgehensweise können die Installationen im laufenden Betrieb erweitert oder aktualisiert werden.

1. Erstellung eines Snapshots der bisherigen Installation.
2. Starten des Betriebssystems des neuen Snapshots auf dem Host oder über einen permanenten NBD-export (ohne COW). Dazu empfiehlt sich eine temporäre TFTP-Konfiguration auf eine bestimmte MAC-Adresse eines zur Wartung zu verwendenden Rechners -> Anpassung der TFTP- und NBD-Konfiguration.
3. Sobald die Konfigurationen und Tests erfolgreich abgeschlossen wurden kann das Image für den allgemeinen Zugriff freigegeben werden -> Anpassung der TFTP-Konfiguration.
4. Sollten während des Betriebs Probleme festgestellt werden, so kann die TFTP-Konfiguration auf das bisherige Image zurückgesetzt werden.

## 9 Zusammenfassung

Durch minimalen Konfigurationsaufwand können alle unter (2) gesetzten Ziele erfüllt werden.

Bedingt durch die Emulation der Hardware ist die Performanz eingeschränkt, für Anwendungen welche 3D-Grafikunterstützung erwarten ist das Setup nicht geeignet. Die CPU- und IO-Performanz vermindert sich, reicht aber für weniger anspruchsvolle Anwendungen wie Büro-, Entwicklungs- und Kommunikationssoftware aus.

### Literatur

- [1] *AnyOS*. Webseite, online verfügbar unter: <http://anyos.sourceforge.net/>. Aufgerufen am 18.01.2012.
- [2] *KMUX*. Webseite, online verfügbar unter: <http://kmux.sourceforge.net/>. Aufgerufen am 18.01.2012.
- [3] WIKIPEDIA: *Thin-Client*. Webseite, online verfügbar unter: [http://de.wikipedia.org/wiki/Thin\\_Client](http://de.wikipedia.org/wiki/Thin_Client). Aufgerufen am 18.01.2012.
- [4] WIKIPEDIA: *Rich-Client*. Webseite, online verfügbar unter: [http://en.wikipedia.org/wiki/Rich\\_client](http://en.wikipedia.org/wiki/Rich_client). Aufgerufen am 18.01.2012.
- [5] *KVM*. Webseite, online verfügbar unter: [http://www.linux-kvm.org/page/Main\\_Page](http://www.linux-kvm.org/page/Main_Page). Aufgerufen am 18.01.2012.
- [6] *qemu*. Webseite, online verfügbar unter: [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page). Aufgerufen am 18.01.2012.
- [7] WIKIPEDIA: *Copy-On-Write*. Webseite, online verfügbar unter: <http://de.wikipedia.org/wiki/Copy-On-Write>. Aufgerufen am 18.01.2012.
- [8] *Upstart*. Webseite, online verfügbar unter: <http://upstart.ubuntu.com/>. Aufgerufen am 18.01.2012.
- [9] JUNJIRO R. OKAJIMA: *aufs*. Webseite, online verfügbar unter: <http://aufs.sourceforge.net/>. Aufgerufen am 18.01.2012.
- [10] *openldap*. Webseite, online verfügbar unter: <http://www.openldap.org/>. Aufgerufen am 18.01.2012.
- [11] *Samba*. Webseite, online verfügbar unter: <http://www.samba.org/>. Aufgerufen am 18.01.2012.
- [12] *Intel PRO/1000 and Gigabit Network Adapters*. Download-Link, online verfügbar: <http://www.intel.com/support/network/sb/cs-006120.htm>. Aufgerufen am 18.01.2012.

# Bitcoin

**Jens Kubieziel**

jens@kubieziel.de

<http://kubieziel.de/>

Das folgende Dokument bietet zunächst eine kurze Übersicht zu Geldfunktionen sowie zu elektronischem Geld. Hauptsächlich behandelt es die Währung Bitcoin. Dabei wird die Funktionsweise sowie die Eigenschaften betrachtet.

## 1 Was ist Geld?

Im Zentrum dieses Artikels steht elektronisches Geld und speziell die Ausprägung Bitcoin. Bevor wir tiefer einsteigen, soll rekapituliert werden, was Geld eigentlich ist und welche Funktionen es hat. Eine tiefere Einführung in die Geldtheorie bietet unter anderem das Buch von Otmar Issing ([Issing06]).

Jeder geht täglich mit Geld in verschiedener Form um. Die Antwort, auf die Frage, was Geld eigentlich ist, wird zumeist mit *Zahlungsmittel* beantwortet. Das ist eine der wichtigsten Funktionen. Denn statt Ware für Ware auszutauschen, wird Ware für Geld und Geld für Ware eingetauscht. Damit umgehen die Akteure verschiedene Probleme. In einer kleinen Beispielwelt leben beispielsweise drei Akteure, Alice, Bob und Charlie. Alle bieten eine Ware an und suchen eine andere Ware. Die Tabelle 1 zeigt, was jeder Akteur bietet bzw. sucht.

	Alice	Bob	Charlie
Angebot	GNU/Linux-Kurs	Kartoffeln	Fahrräder
Nachfrage	Fahrräder	GNU/Linux-Kurs	Kartoffeln

Tabelle 1: Beispielwelt mit Angebot und Nachfrage

In dieser Beispielwelt findet sich kein Akteur, der das anbietet, was ein anderer sucht. Also kommt es nicht zum Tausch. Erst wenn sich jemand entschließt, »um die Ecke« zu tauschen, können Geschäfte stattfinden. Mit der Einführung von Geld in dieser Welt erleichtert sich einiges. Jeder Teilnehmer gibt oder nimmt Geld als Zwischenstation an und somit können Geschäfte stattfinden.

Im obigen Beispiel ergibt sich ein weiteres Problem. Stellen wir uns vor, Bob tauscht seine Kartoffeln mit Alice. Sie verbraucht diese jedoch zu langsam. So werden die Kartoffeln zuerst schrumpelig und sind später ungenießbar, also wertlos.

Mit Geld als Zwischenstück hätte er eine kleinere Menge Kartoffeln kaufen können. Später hätte er wiederum gegen Tausch von etwas Geld eine kleine Menge Kartoffeln kaufen können. Das Geld hat hier also eine *Wertaufbewahrungsfunktion*. In der Vergangenheit wurden daher immer nicht verderbliche Sachen<sup>1</sup> als Geld verwendet.

Nun haben Sie sich vielleicht die Frage gestellt, warum überhaupt Waren oder Dienstleistungen gegen Geldmünzen oder -scheine getauscht werden sollen. Immerhin liefert im obigen Beispiel Charlie ein hochwertiges Fahrrad und bekommt von seinem Kunden als Gegenleistung eine Packung Papier oder einen Sack voller Münzen.

Anfangs bestand die Sicherung in den Münzen selbst. Jede Münze bestand aus einem Metall und war durch dieses gedeckt. Derartiges wird als *Kurantmünzen* bezeichnet. Dies ist heutzutage nicht mehr üblich.<sup>2</sup> In Deutschland war Kurantgeld bis etwa vor 100 Jahren in Umlauf. Es wurde durch so genanntes *Fiatgeld* verdrängt. Hierbei verleiht der Staat dem Geld den Status eines gesetzlichen Zahlungsmittels. Beispielsweise sind Euroscheine gesetzliches Zahlungsmittel. Bei Euromünzen gibt es keine Verpflichtung, mehr als 50 Münzen pro Zahlung anzunehmen ([EG-VO974]). Es gibt keine Hinterlegung von Gold oder anderen Wertgegenständen.

Im täglichen Umgang wird meist zwischen *Bar-* und *Buchgeld* unterschieden. Bargeld sind dabei die Scheine und Münzen. Hingegen ist Buchgeld alles das, was auf diversen Konten gelagert ist. Elektronisches Geld ist eine Ausprägung von Buchgeld.

## 2 Elektronisches Geld

Innerhalb der Europäischen Union existiert seit mehr als zehn Jahren eine feste Definition zu elektronischem Geld. In der Richtlinie 2009/110/EG des Europäischen Parlaments und des Rates vom 16. September 2009 über die Aufnahme, Ausübung und Beaufsichtigung der Tätigkeit von E-Geld-Instituten([EGeld09]) wurde in Artikel 1 Nummer 5 festgelegt:

2. "E-Geld" jeden elektronisch — darunter auch magnetisch — gespeicherten monetären Wert in Form einer Forderung gegenüber dem Emittenten, der gegen Zahlung eines Geldbetrages ausgestellt wird, um damit Zahlungsvorgänge im Sinne des Artikels 4 Nummer 5 der Richtlinie 2007/64/EG durchzuführen, und der auch von anderen

<sup>1</sup> Meist waren das Metalle, wie Gold, Silber oder Kupfer.

<sup>2</sup> Je nach Materialpreisen sind Ein-Cent-Münzen weniger als das Material wert und damit »gedeckt«. Dies ist jedoch eher ein ungewollter Nebeneffekt.

natürlichen oder juristischen Personen als dem E-Geld-Emittenten angenommen wird;

Damit zählt beispielsweise Geld auf einer Kredit- oder Debitkarte nicht als elektronisches Geld, im Gegensatz zu einer GeldKarte. Dabei wird auf dem Chip einer Karte ein Geldbetrag gespeichert. Wenn damit bezahlt wird, so reduziert sich der verfügbare Betrag auf dem Chip. Ein ähnliches System wird in Thüringen bei den so genannten THOSKA-Karten<sup>3</sup> eingesetzt, um Essen in der Mensa oder Kopierkosten zu zahlen. Diese Erscheinungsform elektronischen Geldes trägt den Namen *Kartengeld*. Demgegenüber steht das *Netzgeld*. Wie der Name vermuten lässt, erfolgt die Bezahlung über ein Computernetz. Das Geld selbst ist meist in Form von Bits und Bytes auf einem Datenträger repräsentiert.

In Deutschland war ecash als eine Form elektronischen Geldes populär. Insbesondere in Europa wurde diese Erfindung von einigen Banken unterstützt. Das System stammte von dem amerikanischen Kryptographen David Chaum ([Chaum83]) und setzte auf kryptographischen Erkenntnissen auf. Um elektronisches Geld nutzen zu können, erstellt der Anwender zunächst einen »Geldschein«. Dieses Dokument ist verschlüsselt und niemand kann den Inhalt lesen. Daraufhin begibt der Anwender sich zur Bank und verlangt eine Unterschrift. Die Bank möchte natürlich gern den Wert auf dem Dokument prüfen. Aber aufgrund der Verschlüsselung ist dies nicht möglich. An dieser Stelle kommt nun ein kleiner Trick ins Spiel. Statt eines Scheines erstellt der Anwender 100 Scheine. Davon wählt die Bank zufällig 99 aus und verlangt den Schlüssel zur Entschlüsselung. Wenn alle diese die vom Anwender genannten Eigenschaften haben, geht die Bank davon aus, dass das auch beim letzten verbliebenen der Fall ist. Dieser Schein wird unbesehen unterschrieben. Der Anwender entschlüsselt anschließend den Schein wieder. Jetzt kann er ihn zusammen mit der Unterschrift der Bank zur Bezahlung verwenden. Die Unterschrift garantiert dem Händler, dass er auch wirklich den Gegenwert zurück erhält.<sup>4</sup>

Trotz der tatkräftigen Unterstützung der Banken war dem System kein Erfolg beschieden. Die hinter ecash stehende Firma DigiCash Inc. musste Ende der 90er Jahre Zahlungsunfähigkeit anmelden. Später gingen die Überreste in anderen Firmen auf. Heute spielt ecash im Zahlungsverkehr keine Rolle mehr. Dennoch gab es in den letzten Jahren immer wieder Ansätze das Netzgeld wiederzubeleben.<sup>5</sup> Vielen war dasselbe Schicksal beschieden, wie ecash. Bitcoin zog im Jahr 2011 große Aufmerksamkeit auf sich. Viele Nutzer stürzten sich auf diesen Ansatz. Einige investierten sogar ihr komplettes Vermögen ([Falkvinge11]) in Bitcoins. Doch was ist das und wo liegt der Unterschied zu anderen Ansätzen?

---

<sup>3</sup><http://www.uni-jena.de/thoska.html>

<sup>4</sup>Die Beschreibung lässt die genauen kryptographischen Eigenschaften weg. Details können in der Veröffentlichung von Chaum nachgelesen werden.

<sup>5</sup>Zusammenstellung einiger Aspekte von ecash und vergleichbaren Systemen: <http://www.cs.ut.ee/~lipmaa/crypto/link/protocols/ecash.php>

### 3 Bitcoin

Bitcoin entstammt einer Idee von Satoshi Nakamoto. Er stellte sein Projekt im November 2008 auf einer Mailingliste für Kryptographie vor ([Nakamoto08]). Anfang 2009 folgte die Software ([Nakamoto09]) und es wurde der erste Block Bitcoins herausgegeben.<sup>6</sup>

Bitcoin bezeichnet sich als dezentrale, digitale Währung auf Peer-to-Peer-Basis. Das Geld wird von den Teilnehmer selbst erzeugt und wird zwischen beliebigen Knoten im Netzwerk übertragen. Jede Transaktion wird per Broadcast verteilt und innerhalb weniger Minuten bestätigt. Die Seite [bitcoinstats.org](http://bitcoinstats.org) zeigt die Dauer der letzten Transaktionen an. Durch die Ausschaltung von Zwischenstellen<sup>7</sup> sinken die Transaktionskosten. Die Seite [WeUseCoins.com](http://WeUseCoins.com) bietet ein anschauliches Video über die Grundfunktionen der Währung.

Seit 2009 wuchs die Popularität von Bitcoin und erreichte im Jahr 2011 einen vorläufigen Höhepunkt. Verschiedene Presseberichte stuften Bitcoin als anonymes Zahlungsmittel ein, dass die Macht habe, Regierungen zu stürzen. Mit dem neuen Ansehen kamen dann diverse Probleme. So erkannten Kriminelle ebenso das Potenzial und stahlen die Geldbörsen. Andere missbrauchten fremde Rechner, um neue Bitcoin zu berechnen. Mittlerweile ist es um die Währung wieder etwas ruhiger geworden und es empfiehlt sich, einen ruhigen, besonnenen Blick auf das System zu werfen.

#### 3.1 Bitcoin installieren und in Betrieb nehmen

Die Bitcoin-Software kann von <http://bitcoin.org/> heruntergeladen werden. Sie unterliegt der MIT-Lizenz und steht für alle gängigen Betriebssysteme zur Verfügung. Für Ubuntu gibt es ein PPA. Nutzer von GNU/Linux sollten die Qt4-Laufzeit-Bibliotheken installiert haben.<sup>8</sup> Wenn die Bibliothek auf Ihrem System vorhanden ist, können Sie das Archiv bei SourceForge herunterladen bzw. das Ubuntu-Paket installieren. Nach der Installation und dem ersten Start des Programms nimmt die Software Kontakt zum Netzwerk auf. Dabei wird versucht, die Transaktionshistorie herunterzuladen. Die Abbildung 1 zeigt den Client beim ersten Download.

Der Download der Transaktionshistorie kann sehr lange dauern. Da jede Transaktion der Vergangenheit gespeichert und übertragen wird, steigt die Größe der Historie. Mit einem DSL-Zugang sollten Sie mehr als eine Stunde Downloadzeit einrechnen. Danach steht Ihnen die Software zur Verfügung.

---

<sup>6</sup><https://blockexplorer.com/b/0>

<sup>7</sup>In der bisherigen Praxis sind das zumeist Banken.

<sup>8</sup>`apt-get install libqtgui4` unter Debian-basierten Systemen

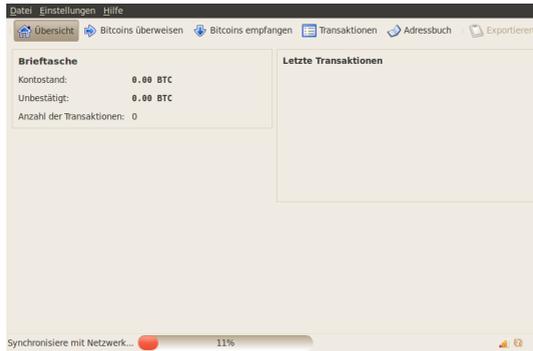


Abbildung 1: Bitcoin-Client während der initialen Synchronisation

Schon als die Software gestartet wurde, hat diese eine Adresse berechnet. Ein Klick auf »Bitcoins empfangen« zeigt Ihnen existierende Adressen an (Abbildung 2).



Abbildung 2: Initiale Adresse des Bitcoin-Clients

Nun wird es Zeit, einige Bitcoins in die Geldbörse zu laden. Einer der Hauptentwickler, Gavin Andresen, vergibt jedem Erstbesucher von FreeBitcoins<sup>9</sup> 0,005 Bitcoins. Das entspricht einem Gegenwert von etwa zwei Cent. Damit haben Sie ein erstes Startkapital. Andere Seiten bieten für Erstbesucher ebenfalls kleine Summen in Bitcoins als Geschenk an. Wenn Sie dies versuchen, so bekommen Sie einen ersten Eindruck über das Programm.

### 3.2 Funktionsweise

Nachdem die ersten Schritte mit der Software gemacht sind, wollen wir einen Blick unter die Motorhaube von Bitcoin werfen. In der Abbildung 2 steht die Bitcoin-Adresse. Diese ist der Zahlungsempfänger, d. h. bei einer Transaktion wird neben der zu überweisenden Summe die Adresse als Empfänger angegeben. Bei der Erzeugung der Adresse kommt ein wenig Kryptographie ins Spiel. Bitcoin nutzt die

<sup>9</sup><https://freebitcoins.appspot.com/>

so genannte Public-Key-Kryptographie. Hierbei sind die Schlüssel zur Ver- und Entschlüsselung unterschiedlich und unabhängig. Das heißt, man kann einen Schlüssel nicht aus einem anderen berechnen. Das bekannteste Beispiel für Public-Key-Kryptographie ist der Algorithmus RSA. Bitcoin setzt auf eine vergleichsweise neue Methode, die Kryptographie mit elliptischen Kurven. Der Unterunterabschnitt 3.3.1 geht genauer darauf ein.

### 3.2.1 Erzeugung der Bitcoin-Adresse

Die digitale Briefftasche heißt im Bitcoin-Jargon *wallet*. Dazu gehört die Datei `wallet.dat`, die sich standardmäßig im Unterverzeichnis `.bitcoin` des Heimatverzeichnisses befindet. Genauso wie die Briefftasche in der realen Welt sollte die `wallet` geschützt sein. Bitcoin bietet an, diese zu verschlüsseln. Weiterhin sollten Sie unbedingt eine oder mehrere Sicherheitskopien dieser Datei anfertigen. Denn im Falle eines Verlusts ist der Inhalt unwiederbringlich verloren.

Anfangs wird ein öffentlicher und ein privater Schlüssel für die digitale Briefftasche erzeugt. Der öffentliche Schlüssel wird über einen so genannten *Fingerprint*, eine eindeutige Prüfsumme, präsentiert. Aus dem Fingerprint und einem zufälligen Hashwert von 32 Bit Größe errechnet sich die Bitcoin-Adresse. Diese wird in einen Wert zur Basis 58 (Base58) umgerechnet. Das heißt, er besteht nur noch aus Groß- und Kleinbuchstaben ohne I, O und l sowie aus Zahlen ohne 0. Der Wert ist in der Abbildung 2 zu sehen und lautet `1DL3JFoKkSVFQgkt3Y2Qx9SyQShw69NXDp`.

### 3.2.2 Transaktion

Nun ist eventuell ein wenig Geld in der digitalen Geldbörse vorhanden. Das will nun in Umlauf gebracht werden. Für eine Transaktion wird eine digitale Signatur aus dem Hashwert der vorhergehenden Transaktion und dem öffentlichen Schlüssels der gerade handelnden Person erzeugt. Diese Transaktion wird dann als Broadcast-Nachricht in das Peer-to-Peer-Netzwerk gesendet. Das Netzwerk markiert diese Nachricht zunächst als *unconfirmed* (unbestätigt). Unbestätigte Transaktionen werden zu einem Block hinzugefügt und das so genannte *Mining* beginnt. Durch zweifache Anwendung des Hashalgorithmus' SHA-256 versucht die Software einen Hashwert zu errechnen, der bestimmten Bedingungen genügt. Sobald dies passiert ist, wird der Block veröffentlicht.

In Unterabschnitt 3.1 wurde erklärt, wie man zu Beginn kostenlos einen kleinen Betrag Bitcoins erhält. Für das hier vorliegende Dokument wurde die Transaktion ebenso durchgeführt. Das heißt, es gibt nun einen Block, der zumindest diese Transaktion enthält. Werfen wir einen Blick darauf. Der Block, der diese Transaktion enthält, trägt die laufende Nummer 162 873 und den Hashwert

000000000000b22169282d02dbd7098b46beaacde5a01a7f2fef0b2d308d79d

Die Seite [BlockExplorer.com](https://blockexplorer.com)<sup>10</sup> listet alle Details der Transaktion<sup>11</sup> auf. Dort ist zu sehen, dass in dem Block insgesamt 13 Transaktionen enthalten sind. Er wurde am 19. Januar 2012 um 9:16:20 Uhr UTC bestätigt und umfasst einen Transaktionswert von 173,68358834 BTC.<sup>12</sup> Das sind knapp 850 €.

Jeder Block ist mit seinem Vorgänger und Nachfolger verbunden und bildet auf diese Weise eine Kette. Den Anfang macht der Block 0.<sup>13</sup> Er wurde am 3. Januar 2009 von Nakamoto berechnet.

### 3.2.3 Mining

Oben wurde bereits kurz auf das Mining eingegangen. Neben der Zusammenfassung von Transaktionen zu einem Block werden beim Mining neue Bitcoins erzeugt. Diese werden dem Erzeuger eines neuen Blocks gutgeschrieben und stehen immer am Anfang des Blocks. Bei der oben angeführten Transaktion wurden die Bitcoins der Adresse 1KUCp7YP5FP8ViRxhfszSUJCTAajK6viGy<sup>14</sup> gutgeschrieben. Die Übersichtsseite zeigt, dass es sich um einen sehr aktiven Anwender handelt. Er hat bereits sehr viele neue Bitcoins erzeugt und an dem Tag allein erschuf er Bitcoins von mehr als 1 000 BTC.

Im Gegensatz zu den heute existierenden Währungen lassen sich Bitcoins nicht unbegrenzt erschaffen. Für die ersten 210 000 Blocks erfolgt eine Gutschrift von 50 Bitcoins. Danach wird der Betrag halbiert. Für jede weiteren 210 000 Blocks erfolgt wiederum eine Halbierung des Betrages. Damit werden nach und nach weniger Bitcoins gutgeschrieben, bis die Gutschrift irgendwann bei Null liegt. Der Zeitpunkt liegt im Jahr 2033. Bis dahin sind 21 Millionen Bitcoins im Umlauf. Die ist die maximale Geldmenge in diesem System. Die Abbildung 3 zeigt den Verlauf der Geldschöpfung im Bitcoin-System.<sup>15</sup>

Das Mining neuer Bitcoins war lange Zeit ein beliebter »Sport«. Viele Nutzer bauten sich spezielle Rechner mit Grafikchips bzw. nutzten spezielle Schaltkreise mit programmierbarer Logik (FPGAs). Als Gegenleistung für die Investition in Technik und Strom erhalten sie Bitcoins. Ab einem bestimmten Umrechnungskurs ist es lohnenswert, die Technik zum Mining einzusetzen. Denn der Wert der 50 BTC ist höher als die entstehenden Stromkosten.

<sup>10</sup><https://blockexplorer.com/>

<sup>11</sup><https://blockexplorer.com/b/162873>

<sup>12</sup>Offizielles Kürzel der Währung Bitcoin.

<sup>13</sup><https://blockexplorer/b/0>

<sup>14</sup><https://blockexplorer.com/a/8gH5JdvcMg>

<sup>15</sup>Die Grafik stammt aus dem Bitcoin-Wiki: [https://en.bitcoin.it/wiki/File:Total\\_bitcoins\\_over\\_time\\_graph.png](https://en.bitcoin.it/wiki/File:Total_bitcoins_over_time_graph.png)

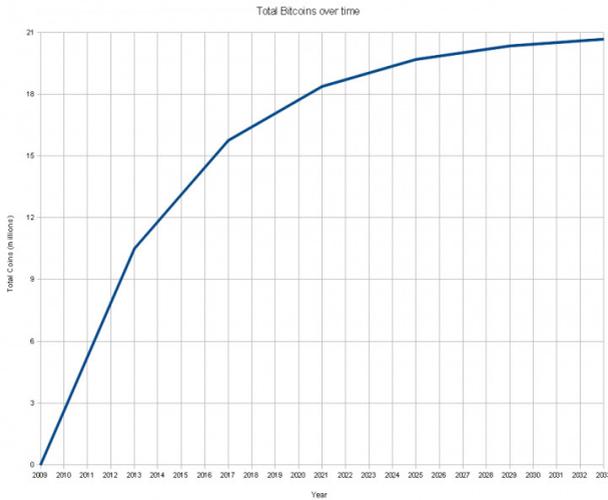


Abbildung 3: Verlauf des Angebots an Bitcoins

Die Stromkosten brachten einzelne Miner in der Vergangenheit in Bedrängnis. In der kanadischen Stadt Mission dürfen spezielle Einheiten Hausdurchsuchungen durchführen, wenn der tägliche Stromverbrauch 93 kWh übersteigt. Dadurch soll der unerlaubte Anbau von Marihuana erkannt werden. Bitcoin-Miner waren hier auch Ziel von Hausdurchsuchungen ([Storm11]). Derzeit ist der Wechselkurs von Bitcoin wieder gesunken und es lohnt sich nicht unmittelbar, das Mining in größerem Ausmaß zu verfolgen.

Nun stellt sich die Frage, was nach der Erschaffung aller Bitcoins passiert. Aus welchem Grund sollten weitere Blöcke berechnet werden? Denn es werden keine neuen Bitcoins geschöpft. Die Antwort liegt hier in Transaktionskosten. Für jede Transaktion zahlt der Nutzer eine kleine Gebühr an den Miner. Dieser nimmt die Gebühren und fasst die Transaktionen zu einem Block zusammen. Somit ist theoretisch das Fortbestehen der Währung gesichert.

### 3.2.4 Bitcoin als Währung

Neben dem Mining besteht eine andere Möglichkeit, Bitcoin zu nutzen. Wie auch bei anderen Währungen lassen sich Bitcoin in Euro, US-Dollar und andere wechseln. Hierzu haben sich verschiedene Handelsplätze herausgebildet. Am bekanntesten dürfte die Seite Mt. Gox<sup>16</sup> sein. Dies ist gleichzeitig die weltgrößte Börse für Bitcoin.

<sup>16</sup><https://mtgox.com/>

Dort stehen momentan 18 Fremdwährungen zum Tausch zur Verfügung. Weitere Börsen sind unter anderem:

- TradeHill.com
- Bitcoin.de sowie
- Liste verschiedener Börsen:  
[https://de.bitcoin.it/wiki/Handel#Exchange-Plattformen\\_.2F\\_elektronische\\_B.C3.B6rsen](https://de.bitcoin.it/wiki/Handel#Exchange-Plattformen_.2F_elektronische_B.C3.B6rsen)

Zumeist funktionieren die Börsen so, dass sich der Nutzer anfangs registriert und ein Konto zugewiesen bekommt. Auf das Konto überweist er einen Betrag in einer Währung. Im folgenden kann er Tauschvorgänge in Bitcoin und zurück vornehmen. Die grundsätzliche Funktionsweise unterscheidet sich nicht von den üblichen Devisenmärkten.

Aber es soll nicht unerwähnt bleiben, dass Mt. Gox wie auch andere Börsen in der Vergangenheit Ziel von Angriffen waren. Bei dem größten Angriff im Juni 2011 wurden Bitcoins im Gegenwert von fast neun Millionen US-Dollar entwendet. Diese Vorfälle zeigten, dass es auch hier sehr wichtig ist, dass sich Nutzer des Risikos bewusst sind und entsprechende Sicherheitsvorkehrungen treffen.

Weiterhin finden sich in diversen Diskussionsforen Beschwerden über verschiedene Handelsstellen, u. a. auch Mt. Gox. Vor einem Geschäft mit der jeweiligen Tauschbörse sollten Sie daher Erkundigungen einholen und im Zweifel eher die Finger davon lassen.

### **3.3 Eigenschaften**

In den folgenden Abschnitten wird auf einige wichtige Eigenschaften und auf die Sicherheit der Währung eingegangen. Dies soll den Ausblick auf die neue Währung abrunden.

#### **3.3.1 Kryptographie**

Bitcoin setzt auf Elliptische-Kurven-Kryptographie (ECC), genauer auf den digitalen Signatur-Algorithmus mit elliptischen Kurven.<sup>17</sup> ECC ist ein asymmetrisches Kryptosystem. Dabei besitzen Sender und Empfänger verschiedene Schlüssel zur Ver- und Entschlüsselung. Dies steht im Gegensatz zu symmetrischen Kryptosystemen, wo der Schlüssel zur Ver- und Entschlüsselung gleich ist. Ein bekanntes

---

<sup>17</sup>Elliptic Curve Digital Signature Algorithmus (ECDSA)

Beispiel für asymmetrische Systeme ist RSA und der Algorithmus AES wird derzeit als symmetrisches Verfahren eingesetzt.

Elliptische Kurven sind, wie schon im Namen steckt, Kurven. Sie erfüllen die Gleichung  $y^2 = x^3 + ax + b$  und nutzen weiterhin einen unendlichen Punkt. Die Abbildung 4 zeigt einige Beispiele für elliptische Kurven. Die Abbildung entstammt aus der englischsprachigen Wikipedia.<sup>18</sup> Es wird festgelegt, wie zwei Punkte der Kurve zu addieren und zu multiplizieren sind. Nach der Festlegung kann mit den Punkten der Kurve wie mit bekannten Zahlen gerechnet werden.<sup>19</sup>

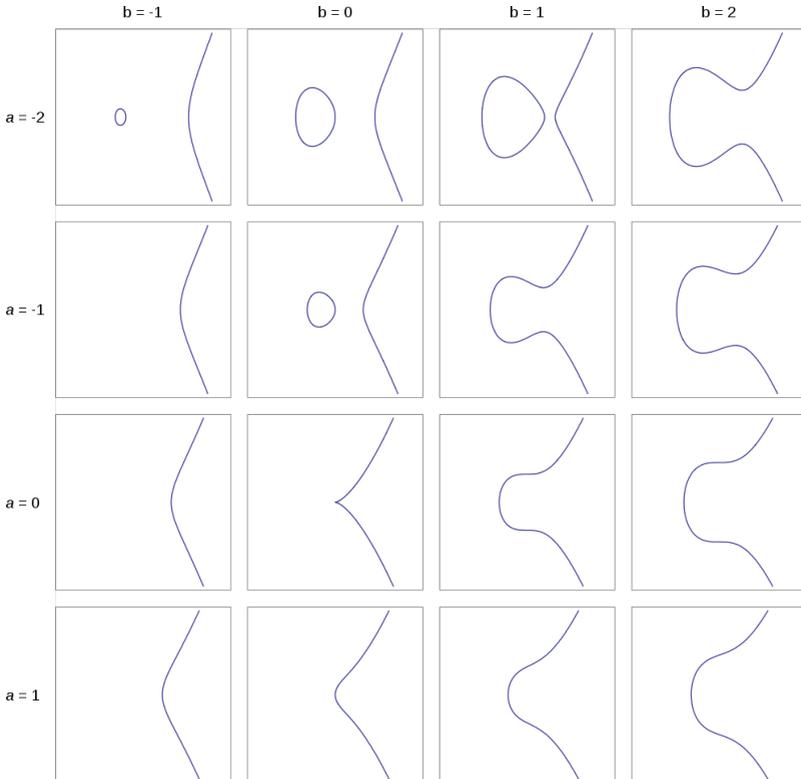


Abbildung 4: Elliptische Kurven für verschiedene Werte von  $a$  und  $b$  (aus der englischsprachigen Wikipedia)

Insbesondere lässt sich mit elliptischen Kurven Kryptographie betreiben.<sup>20</sup> Der

<sup>18</sup><https://en.wikipedia.org/wiki/File:EllipticCurveCatalog.svg>

<sup>19</sup>Genauere Details können Sie dem entsprechenden Artikel zu elliptischen Kurven in der deutschsprachigen Wikipedia entnehmen: [https://de.wikipedia.org/wiki/Elliptische\\_Kurve](https://de.wikipedia.org/wiki/Elliptische_Kurve)

<sup>20</sup>Die Aussagen wurden hier stark vereinfacht. Als Einstieg in die Thematik eignen sich die Artikel in der

Digital Signature Standard wurde in der Publikation FIPS PUB 186-3 ([FIPS186-3]) spezifiziert. Darin findet sich auch ein Algorithmus für ECC. Certicom Research hat verschiedene Parameter für die ECC festgelegt. Bitcoin nutzt den Vorschlag secp256k1. Laut [Certicom00] entspricht das einer Schlüssellänge von 128 Bit bei der AES-Chiffre oder 3 072 Bit bei RSA. Diese Längen werden in Abhängigkeit von der Quelle als sicher bis 2020 ([FNISA10]) oder sogar bis 2090 ([Lenstra04]) angesehen.

Des Weiteren wird SHA-256 als Hashalgorithmus verwendet. Dieser wird derzeit als sicher angesehen und stellt die aktuelle Empfehlung (nebst den Hashfunktionen SHA-224, SHA-384 und SHA-512) dar.

Die verwendete Kryptographie wurde von den Entwicklern gut ausgewählt und kann derzeit als sicher angesehen werden. Bislang fanden Forscher keine Schwachstellen in der Umsetzung in Code. Aber die Zukunft hält hier einige Probleme bereit. Bitcoin hat den Anspruch über einen längeren Zeitraum zu existieren. Irgendwann werden sicher Schwachstellen in den verwendeten Algorithmen gefunden werden. Dann stellt sich für die Entwickler die Frage, wie sie zu einem neuen Algorithmus in sicherer Form wechseln.

### **3.3.2 Rückabwickelbarkeit von Transaktionen**

Wie in Unterunterabschnitt 3.2.1 beschrieben wurde, entsteht eine Bitcoin-Adresse aus dem Schlüssel und anderen Daten. Die persönliche Identität des Schlüsselinhabers spielt keine Rolle. Das heißt, wenn eine Transaktion an eine andere Adresse getätigt wird, ist unter Umständen nicht bekannt, wer sich dahinter verbirgt. Das macht es schwierig, Fehler zu korrigieren. Wenn also eine Transaktion an die falsche Adresse geleitet wurde, gibt es innerhalb von Bitcoin keine Möglichkeit, dies rückabzuwickeln. Weiterhin ist es gegebenenfalls auch nicht möglich, den Empfänger auf anderem Wege zu ermitteln. Also ist das Geld in gewissem Sinne verloren. Banküberweisungen sind hier vergleichbar, wobei der Empfänger einer Überweisung ermittelbar ist. Aber auch bei Überweisungen gibt es seitens der Banken keine Möglichkeit, diese nach Abschluss zurückzuüberweisen.

### **3.3.3 Anonymität der Währung**

Eine der großen Versprechungen mit denen Bitcoin gestartet ist, ist die Anonymität. Schließlich ist der Name des Senders oder Empfängers nirgends gespeichert. Jeder Teilnehmer kann beliebig viele Adressen erzeugen. Auf der anderen Seite ist jede Transaktion in einem Block gespeichert und jeder Block wird an jeden Teilnehmer übertragen. Das klingt nicht besonders anonym.

---

englischsprachigen Wikipedia sowie die dort genannten weiterführenden Links.

Gerade wenn eine Bitcoin-Adresse zu einer Person zuordenbar ist, dann können alle vergangenen Aktivitäten, die zu der Adresse gehören, aufgedeckt werden. Es ist recht einfach, bei Suchmaschinen Bitcoin-Adressen zu finden. So existieren private Blogbetreiber, öffentliche Jabber-Server und sogar gemeinnützige Organisation mit den entsprechenden Adressen. Seiten, wie der oben genannte BlockExplorer, zeigen zu einer Adresse alle Transaktionen an. So lässt sich nachvollziehen, wieviel Spenden beispielsweise jemand erhalten hat. Derartige Informationen können unter anderem für Finanzämter recht wertvoll sein.

Als Empfänger von Bitcoin empfiehlt es sich daher, für jeden Empfang eine separate Bitcoin-Adresse anzulegen. Das Geld wird auf diese Adresse empfangen und von dieser Adresse werden wiederum Ausgaben bezahlt. Sofern nicht Lieferadressen oder andere deanonymisierende Merkmale bekannt werden, sind diese Transaktionen vermutlich anonym. Dennoch gilt, sobald die Bitcoin-Adresse aus irgendeinem Grund einer Person zuordenbar wird, sind auch alle Transaktionen verknüpfbar. Das System ist in dieser Hinsicht also sehr fragil und in keiner Weise mit dem sehr anonymen Bargeld vergleichbar.

Reid und Harrigan untersuchten die Netzwerkstrukturen von Bitcoin[Reid11]. Sie kamen ebenso zu dem Ergebnis, dass sich der Geldfluss innerhalb des Netzwerks gut nachvollziehen lässt. Die Forscher betrachteten zwei Fälle, einen Diebstahl von Bitcoins und Spender von WikiLeaks. In beiden Fällen konnten Sie den Lauf des Geldes sehr gut nachzeichnen.

### **3.4 Zusammenfassung**

Über Bitcoin gibt es sicher noch viel mehr zu schreiben. Das System hat sehr viele positive wie negative Seiten. Um es mit den Worten von Dan Kaminsky, einem Forscher und Experten auf dem Gebiet der IT-Sicherheit, zu sagen:

I have a tremendous amount of respect for BitCoin; I count it in the top five most interesting security projects of the decade.

Insofern lohnt es sich, einen Blick auf die neue Währung zu werfen. Der Erfinder bzw. die aktuellen Entwickler haben sich sehr viele Gedanken zu dieser Währung gemacht und einige Ansätze sind viel versprechend. Vielleicht heißt die neue Währung nicht Bitcoin, aber hoffentlich war Bitcoin eines der Fundamente.

### **Literatur**

[Issing06] Otmar Issing: *Einführung in die Geldtheorie*. München, Vahlen. 14. Auflage, 2006.

- [EG-VO974] Verordnung (EG) Nr. 974/98 des Rates vom 3. Mai 1998 über die Einführung des Euro. Amtsblatt Nr. L 139 vom 11/05/1998 S. 0001–0005.
- [EGeld09] Richtlinie 2009/110/EG des Europäischen Parlaments und des Rates vom 16. September 2009 über die Aufnahme, Ausübung und Beaufsichtigung der Tätigkeit von E-Geld-Instituten, zur Änderung der Richtlinien 2005/60/EG und 2006/48/EG sowie zur Aufhebung der Richtlinie 2000/46/EG. Amtsblatt Nr. L 267 vom 10/10/2009 S. 0007–0017
- [Chaum83] David Chaum: *Blind signatures for untraceable payments*, *Advances in Cryptology – Crypto '82*, Springer-Verlag (1983), 199–203. <http://blog.koehntopp.de/uploads/Chaum.BlindSigForPayment.1982.PDF>
- [Falkvinge11] Rick Falkvinge: *Why I'm putting all my savings into Bitcoin*. Blogeintrag, 2011. <http://falkvinge.net/2011/05/29/why-im-putting-all-my-savings-into-bitcoin/>
- [Nakamoto08] Satoshi Nakamoto: *Bitcoin: A Peer-to-Peer Electronic Cash System*. The Cryptography Mailing List, 2008.
- [Nakamoto09] Satoshi Nakamoto: *Bitcoin v0.1 released*. The Cryptography Mailing List, Januar 2009.
- [Storm11] Darlene Storm: *Bitcoin miners busted? Police confuse bitcoin power usage for pot farm*. Computerworld Blogs, Security Is Sexy, 2011. [http://blogs.computerworld.com/18335/bitcoin\\_miners\\_busted\\_police\\_confuse\\_bitcoin\\_power\\_usage\\_for\\_pot\\_farm](http://blogs.computerworld.com/18335/bitcoin_miners_busted_police_confuse_bitcoin_power_usage_for_pot_farm)
- [FIPS186-3] National Institute of Standards and Technology: *Federal Information Processing Standards (FIPS) Publication 186-3: Digital Signature Standard (DSS)*. Juni 2009. [http://csrc.nist.gov/publications/fips/fips186-3/fips\\_186-3.pdf](http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf)
- [Certicom00] Certicom Research: *SEC 2: Recommended Elliptic Curve Domain Parameters*. Version 2.0, Standards for Efficient Cryptography. September 2009. <http://www.secg.org/download/aid-780/sec1-v2.pdf>
- [FNISA10] Agence nationale de la sécurité des systèmes d'information: *Référentiel Général de Sécurité version 1.0. Annexe B1, Mécanismes cryptographiques*. Januar 2010. [http://www.ssi.gouv.fr/IMG/pdf/RGS\\_B\\_1.pdf](http://www.ssi.gouv.fr/IMG/pdf/RGS_B_1.pdf)
- [Lenstra04] Arjen K. Lenstra: *Key Lengths*. Contribution to The Handbook of Information Security. Juni 2004. [http://www.keylength.com/biblio/Handbook\\_of\\_Information\\_Security\\_-\\_Keylength.pdf](http://www.keylength.com/biblio/Handbook_of_Information_Security_-_Keylength.pdf)
- [Reid11] Fergal Reid, Martin Harrigan: *An Analysis of Anonymity in the Bitcoin System*. First International Workshop on Security and Privacy in Social Networks 2011. Juni 2011. <http://anonymity-in-bitcoin.blogspot.com/>



# Busverkehr unter Linux

**Uwe Berger**

bergeruw@gmx.net

Wie allgemein bekannt, entwickelt Grüner Tee das beste Aroma, wenn er mit 70°C heißen Wasser aufgegossen wird und dann idealerweise 90 Sekunden zieht.<sup>1</sup> Wie steuert man die optimale Teezubereitung mit einem Linux-PC? Die dazu notwendigen Grundlagen soll dieser Aufsatz vermitteln.

## 1 Messen, Steuern und Regeln

Zugegeben, der Teekenner wird die Zubereitung seines Lieblingsgetränkes nicht unbedingt mit Hilfe eines Computers durchführen, aber dieses einfache Beispiel ist typisch für ein übliches Problem der Mess- und Regelungstechnik. Es ist ein Sollwert (hier 70°C) zu erreichen und solange zu halten, bis der Tee aufgegossen wird.

Damit dies funktioniert, werden Sensoren zur Ermittlung der momentanen Temperatur sowie elektronisch ansteuerbare Schalter zum Ein- und Ausschalten der Heizelemente benötigt. Wenn die Prozessregelung mit Hilfe eines PC erfolgt, sollten diese Komponenten kompatibel zu den üblichen I/O-Schnittstellen sein. Ideal wären standardisierte Schnittstellen, welche auch unter Linux ansteuerbar sind und für die es diverse Peripheriemodule gibt.

## 2 Busverkehr

Eine solche Schnittstelle stellen beispielsweise standardisierte Bussysteme<sup>2</sup> dar. Hier sind über ein Übertragungsmedium (z.B. ein oder mehrere physikalische Leitung) die verschiedensten Teilnehmer miteinander verbunden. Auf die Physik aufsetzend

---

1 [http://de.wikipedia.org/wiki/Gr%C3%BCner\\_Tee#Zubereitung](http://de.wikipedia.org/wiki/Gr%C3%BCner_Tee#Zubereitung)

2 nebenbei, „Bus“ ist die Abkürzung zu „Binary Unit System“

ist jeweils ein einheitliches Übertragungsprotokoll definiert, welches alle verbundenen Teilnehmer verstehen und hauptsächlich folgendes festlegt:

- zeitliche Definition der Datenübertragung
- Hierarchie der Busteilnehmer (Master/Slave) sowie deren Adressierung
- Datenrichtung (uni- oder bidirektional)

Betrachtet man nun die Anschlüsse eines PCs, sind bereits einige Schnittstellen vorhanden, die das Ausgangsproblem (Ein- und Ausgabe von Daten) lösen könnten. Aber diese Schnittstellen sind meist zu universell ausgelegt, sodass sie für diese sehr einfachen Aufgaben ungeeignet sind oder die Komponenten zu teuer werden würden. Beispielsweise kann an einem USB-Port in der Regel nur ein Teilnehmer angeschlossen werden. Andere Schnittstellen sind wiederum zu spezialisiert (z.B. VGA, HDMI). Aus diesem Grund werden in der Folge zwei einfache und weit verbreitete Bussysteme beschrieben.

## 2.1 1-Wire Bus

### Theorie

Bei dem ersten zu besprechenden System handelt es sich um einen seriellen Datenbus, der von der Firma Dallas Semiconductor Corp.<sup>3</sup> (heute ein Teil der Firma Maxim<sup>4</sup>) entwickelt wurde. Die Bezeichnung 1-Wire- bzw. One-Wire-Bus (Eindraht-Bus) ist der Tatsache geschuldet, dass die Kommunikation über nur eine Datenleitung erfolgt. Des Weiteren ist eine Masseleitung erforderlich.

Die Datenübertragung erfolgt dabei asynchron, bidirektional und in Blöcken von je 64 Bit. Jeder Slave hat eine eindeutige 64-Bit-Adresse, welche sich aus einem 8-Bit-Typ-Code, einer 48 Bit langen Seriennummer und einer Checksumme von 8 Bit zusammensetzt. Wegen dieser Eindeutigkeit der Adresse ist es theoretisch möglich beliebig viele Slaves (auch gleichen Typs) an den Bus anzuschließen. Grenzen werden allerdings durch die Physik, also zeitliches Verhalten, Leitungsqualität, -länge usw., gesetzt. Im Idealfall sind teilweise Leitungslängen von mehreren 100 m erreichbar. Die Vorgehensweise beim Einsatz von 1-Wire-Komponenten ist hauptsächlich in Form von Application-Notes<sup>5</sup> der Firma Maxim beschrieben.

Ein interessanter Aspekt ist, dass die Stromversorgung der 1-Wire-Slaves über die Datenleitung erfolgt ("Parasitic Power"). Bei inaktiver Kommunikation wird dabei ein Speicherkondensator aufgeladen, der in jedem 1-Wire-Slave vorhanden ist. Wäh-

---

3 <http://www.maxim-ic.com/company/dallas/>

4 <http://www.maxim-ic.com/>

5 [http://www.maxim-ic.com/an\\_procline2.cfm/procline/21](http://www.maxim-ic.com/an_procline2.cfm/procline/21)

rend der Kommunikation erfolgt die Stromversorgung aus diesem Kondensator. Reicht dies nicht aus, besteht bei vielen 1-Wire-Bausteinen die Möglichkeit die Stromversorgung über eine zusätzliche Leitung zu realisieren.

Verschiedener US-Patente der Firma Maxim schränken die Implementierung von Slaves „in Software“ ein. Aus diesem Grund sind 1-Wire-Komponenten fast ausschließlich nur von dieser Firma erhältlich. Das Spektrum<sup>6</sup> reicht von einfachen Sensoren, Speicher-, RTC<sup>7</sup>-Bausteinen bis zu elektronischen Schaltern u.ä.. Die iButton-Technologie<sup>8</sup> setzt auf den 1-Wire-Standard auf.

## Anschluss an den PC

Um 1-Wire-Baugruppen an einen PC anschließen zu können, wird ein Bus-Master benötigt, der die Kommunikation zwischen den Teilnehmern (Slaves) regelt. Auf den Webseiten des OWFS-Projektes<sup>9</sup> findet man dazu einen kleinen Überblick<sup>10</sup> zu den verschiedenen Möglichkeiten unter Linux.

Als erstes sind dabei einfache Schaltungen<sup>11</sup> zu erwähnen, die teilweise nur wenige passive Bauelementen beinhalten und sich somit auch als Selbstbauprojekt eignen. Diese Schaltungen werden dann z.B. an die serielle Schnittstelle des PC angeschlossen und entsprechende Software realisiert die Funktionalität des Bus-Masters. Meist beschränken sich diese Lösungen aber auf nur wenige unterschiedliche 1-Wire-Bausteintypen. Elektrisch ungünstiger Aufbau und Bauteilstreuungen erschweren u.U. die Fehlersuche im Problemfall.

Einfacher und sicherer ist die Verwendung spezieller Bus-Master-ICs der Firma Maxim, die als Einzelbauteile erhältlich oder in fertigen kommerziellen Lösungen zu finden sind. Der Anschluss an den PC erfolgt dabei meist über die USB- oder serielle Schnittstelle. Einige Bus-Master-ICs ermöglichen sogar das Kombinieren von 1-Wire- und I<sup>2</sup>C-Bussystemen.

Für die anschließenden Beispiele wurde ein USB-1-Wire-Adapter DS9490R<sup>12</sup> auf Basis eines Maxim-ICs DS2490 verwendet. Dieser IC wird seit geraumer Zeit durch das Kernel-Modul `ds2490` unterstützt, welches wiederum auf das Modul `wire` aufsetzt.

## Software-Praxis

Nach dem Anschluss der oben genannten Hardware am USB-Port und dem Laden

---

6 <http://www.maxim-ic.com/products/1-wire/>

7 Real Time Clock

8 <http://www.maxim-ic.com/products/ibutton/index.cfm>

9 <http://owfs.org>

10 <http://owfs.org/index.php?page=bus-masters>

11 <http://owfs.org/index.php?page=com-ds9097-passive>

12 Beziehbar z.B. über <http://www.fuchs-shop.com>

der entsprechenden Kernel-Module sollten u.a. folgende Meldungen mit `dmesg` sichtbar werden:

```
# dmesg
...
Driver for 1-wire Dallas network protocol.
usbcore: registered new interface driver DS9490R
wl_master_driver wl bus master: Family 10 for 10.000801b55f40.b9 is ...
wl_master_driver wl bus master: Family 10 for 10.000801b56e7d.57 is ...
wl_master_driver wl bus master: Family 81 for 81.0000002e831d.5b is ...
wl_master_driver wl bus master: Family 29 for 29.000000089505.b1 is ...
```

Es wurden vier 1-Wire-Komponenten gefunden. Anhand der Familien-Kennung<sup>13</sup> ist ermittelbar, um welche konkreten Typen es sich handelt:

- Family 10: Temperatursensor DS18S20
- Family 29: 8-Kanal-Schalter DS2408
- Family 81: serieller ID-Button (vermutlich im DS9490R integriert)

Um nun mit diesen 1-Wire-Slaves Daten austauschen zu können, bietet sich das One-Wire-Filesystem<sup>14</sup> (OWFS) an. Aufsetzend auf das `fuse`-Kernel-Modul<sup>15</sup>, wird hier für jeden Slave im 1-Wire-Bus ein virtuelles Verzeichnis generiert. Dessen weitere Unterstruktur richtet sich nach dem Typ des Bausteins.

Es empfiehlt sich OWFS aus dem Quellen zu übersetzen und zu installieren. Nach dem Start von `owfs` (konkrete Startparameter, siehe Manpage):

```
# /opt/owfs/bin/owfs -u -m /home/owtest/mnt/ow -s 3000
```

bietet sich, ab Mountpoint `/home/owtest/mnt/ow` u.a. folgendes Bild:

```
# ls /home/owtest/mnt/ow/ -al
...
drwxrwxrwx 1 root    root      8 Jan  9 20:48 10.405FB5010800
drwxrwxrwx 1 root    root      8 Jan  9 20:48 10.7D6EB5010800
drwxrwxrwx 1 root    root      8 Jan  9 20:48 29.059508000000
drwxrwxrwx 1 root    root      8 Jan  9 20:48 81.1D832E000000
...
```

Für jedes gefundene 1-Wire-Modul wurde ein Unterverzeichnis angelegt, dessen Name aus der eindeutigen Adresse (siehe oben) gebildet wurde. Im Fall eines Temperatursensors DS18S20 (Familie 10) sieht die weitere Verzeichnisstruktur wie folgt aus:

---

<sup>13</sup> <http://owfs.sourceforge.net/family.html>

<sup>14</sup> <http://owfs.org>

<sup>15</sup> <http://fuse.sourceforge.net/>

```

# tree /home/owtest/mnt/ow/10.405FB5010800/
/home/owtest/mnt/ow/10.405FB5010800/
|-- address
|-- alias
|-- crc8
|-- errata
|   |-- die
|   |-- trim
|   |-- trimblanket
|   |-- trimvalid
|-- family
|-- id
|-- locator
|-- power
|-- r_address
|-- r_id
|-- r_locator
|-- temperature
|-- temphigh
|-- templow
\-- type

```

Über diese Unterverzeichnisse kann nun mit dem 1-Wire-Bausteinen kommuniziert werden. Um z.B. den Temperaturwert eines DS18S20 auszulesen, reicht ein einfaches `cat`:

```

# cat /home/owtest/mnt/ow/10.405FB5010800/temperature
6.4375

```

Die Temperatur wird dabei in °C ausgegeben.

Analog zum Lesen erfolgt auch das Senden von Daten zum Baustein über die entsprechenden Unterverzeichnisse. Hier ein Beispiel mit dem Kommando `echo`:

```

# cat /home/owtest/mnt/ow/10.7D6EB5010800/temphigh
0
# echo "42" > /home/owtest/mnt/ow/10.7D6EB5010800/temphigh
# cat /home/owtest/mnt/ow/10.7D6EB5010800/temphigh
42

```

Somit ist es kein Problem, mit den üblichen Dateizugriffsroutinen der jeweiligen bevorzugten Programmiersprache (z.B. C: `fopen`, `fread`, `fwrite`, `fclose`) auf den 1-Wire-Bus zuzugreifen.

Neben dem „reinen“ Filesystem bietet OWFS noch weitere Möglichkeiten die Slaves eines 1-Wire-Bus anzusprechen:

- `owhttpd`: schreibender/lesender Zugriff über eine rudimentäre Weboberfläche

- `owftpd`: Zugriff mittels der üblichen FTP-Kommandos `put` (Schreiben) und `get` (Lesen)
- `owserver`: schreibender/lesender Zugriff via TCP/IP

Zusätzlich zu den OWFS-Kommandozeilentools `owdir`, `owread`, `owrite` und `owpresent` werden native Zugriffs-Bibliotheken für Tcl, Perl sowie PHP zur Verfügung gestellt. Für den ambitionierten Programmierer öffnet sich eine große Spielwiese.

Eine weitere Alternative zur Datenkommunikation mit 1-Wire-Bausteinen stellt das Programm `DigiTemp`<sup>16</sup> dar. Das Programm kommt ohne spezielle Kernel-Module aus und ist auch für DOS/Windows verfügbar. Es werden allerdings nicht alle bekannten 1-Wire-Module abgedeckt. Eine entsprechende Übersicht ist auf der Projektseite zu finden.

## 2.2 I<sup>2</sup>C-Bus

### Theorie

Bei dem, von der Firma Philips Semiconductors<sup>17</sup> entwickelten I<sup>2</sup>C-Bus<sup>18</sup> (I<sup>2</sup>C Abkürzung für Inter Integrated Circuit (IIC)) handelt es sich um eine synchrone, serielle, bidirektionale Datenverbindung, welche über zwei physikalische Leitungen realisiert wird. Er wurde ursprünglich zur internen Kommunikation zwischen verschiedenen Baugruppen eines Gerätes konzipiert. Mittlerweile hat sich I<sup>2</sup>C als ein Industriestandard für Steuerungs- und Überwachungslösungen mit kurzen Buslängen etabliert. Aus lizenzrechtlichen Gründen findet man bei einigen anderen Herstellern die Bezeichnung TWI-Bus (Two-Wire Interface). Einige Hersteller verwenden die I<sup>2</sup>C-Spezifikation als Ausgangsbasis für Eigenentwicklungen. Erwähnenswert dabei ist z.B. Intels<sup>19</sup> SMBus<sup>20</sup>, der auf zahlreichen Mainboards eingesetzt wird. Viele ICs und Baugruppen unterstützen beide Standards.

Über die beiden Signalleitungen (Taktleitung: SCL; Datenleitung: SDA) können, je nach eingestellten Modus und verwendeter Hardware, Übertragungsraten von bis zu 3,4 Mbit/s erreicht werden. Dabei gibt ein Bus-Master den Takt über SCL vor (deshalb synchrone Verbindung) und sendet Daten über SDA an den jeweils adressierten Slave (7-Bit-Adresse), welcher wiederum antworten darf. Zusätzlich sind noch zwei Leitungen für die Stromversorgung der Busteilnehmer erforderlich. Es dürfen bis zu 127 Teilnehmer an dem Bus angeschlossen sein, Multimaster-Mode ist

<sup>16</sup> <http://www.digitemp.com/>

<sup>17</sup> heute NXP Semiconductors: <http://www.nxp.com>

<sup>18</sup> <http://www.i2c-bus.org>

<sup>19</sup> <http://www.intel.com>

<sup>20</sup> <http://www.smbus.org/>

zulässig. Die maximal erreichbare Buslänge beschränkt sich auf nur wenige Meter. Eine ausführliche Spezifikation findet man im Datenblatt<sup>21</sup>.

Wegen der weiten Verbreitung dieses Busstandards werden sehr viele unterschiedliche I<sup>2</sup>C-Komponenten von unterschiedlichen Herstellern angeboten. Das Angebot reicht von ICs<sup>22</sup> (z.B. Sensoren, RTC, Speicher, Expander) bis hin zu komplexen Baugruppen zur Ansteuerung von Grafikkarten, Audiosystemen usw..

## Anschluss an den PC

Es stellt sich nun die Frage, wie man einen I<sup>2</sup>C-Bus an einen Standard-PC anschließt. Prinzipiell geht es hier auch wieder um einen Bus-Master, der die Datenkommunikation zu den angeschlossenen Slaves abbilden kann.

Ganz verwegene Bastler suchen den I<sup>2</sup>C-Anschluß, der auf den meisten Motherboards vorhanden ist. Dieser ist meist als Stiftleiste oder zu mindestens als Lötungen ausgeführt ist. Dazu sollte das entsprechende Datenblatt des jeweiligen Motherboard konsultiert werden.

Einfacher und sicherer ist es, einen Adapter für eine der vorhandenen externen PC-Schnittstellen zu verwenden:

- fertige kommerzielle Lösungen (z.B. I<sup>2</sup>C-USB-Dongle<sup>23</sup>), bei denen man aber auf einen, nicht immer gegebenen, Linux-Support angewiesen ist bzw. diesen selbst implementieren muss
- eine der vielen Selbstbaulösungen, z.B.:
  - USB: `i2c-tiny-usb`<sup>24</sup> für das es seit Kernelversion 2.6.22 ein Kernel-Modul (`i2c-tiny-usb`) gibt
  - Parallel-Schnittstelle: diverse mehr oder weniger<sup>25</sup> komplizierte Schaltungen für das Kernel-Modul `i2c-parport`

Für die weiteren Erläuterungen wurde ein I<sup>2</sup>C-Adapter auf Basis des BLIT2008-Board<sup>26</sup> verwendet, welches, die entsprechenden Firmware<sup>27</sup> vorausgesetzt, ebenfalls mit dem Kernel-Modul `i2c-tiny-usb` zusammenarbeitet.

## Software-Praxis

Nach dem Anschluss der Hardware sollten die Kernel-Module `i2c-dev` und (in

---

21 [http://www.nxp.com/acrobat/usermanuals/UM10204\\_3.pdf](http://www.nxp.com/acrobat/usermanuals/UM10204_3.pdf)

22 [http://www.rm-wissen.de/index.php/I2C\\_Chip-%C3%9Cbersicht](http://www.rm-wissen.de/index.php/I2C_Chip-%C3%9Cbersicht)

23 <http://www.codemercs.com/index.php?id=253&L=0>

24 [http://www.harbaum.org/till/i2c\\_tiny\\_usb/index.shtml](http://www.harbaum.org/till/i2c_tiny_usb/index.shtml)

25 <http://www.mjmwired.net/kernel/Documentation/i2c/busses/i2c-parport>

26 <http://bralug.de/wiki/BLIT2008-Board>

27 [http://bralug.de/wiki/BLIT2008-Board\\_mit\\_i2c-tiny-usb-Firmware](http://bralug.de/wiki/BLIT2008-Board_mit_i2c-tiny-usb-Firmware)

unserem Fall) `i2c-tiny-usb` geladen sein. Mit `dmesg` erscheint ungefähr dies:

```
# dmesg
...
i2c-tiny-usb 1-4.4:1.0: version 2.05 found at bus 001 address 009
i2c i2c-2: connected i2c-tiny-usb device
usbcore: registered new interface driver i2c-tiny-usb
```

Dabei werden für jeden gefundenen I<sup>2</sup>C-Bus entsprechende Devices angelegt:

```
# ls /dev/i2c* -al
crw-rw---- 1 root root 89, 0 2012-01-05 15:07 /dev/i2c-0
crw-rw---- 1 root root 89, 1 2012-01-05 15:07 /dev/i2c-1
crw-rw---- 1 root root 89, 2 2012-01-05 15:07 /dev/i2c-2
```

Verwendet man das Programm `i2cdetect`<sup>28</sup> der I<sup>2</sup>C-Tools<sup>29</sup> für Linux, erhält man ein paar detailliertere Informationen zu den Bussen:

```
# i2cdetect -l
i2c-0  smbus          SMBus nForce2 adapter at 1c00          SMBus adapter
i2c-1  smbus          SMBus nForce2 adapter at 1c40          SMBus adapter
i2c-2  i2c              i2c-tiny-usb at bus 001 device 007     I2C adapter
```

Man sieht zwei interne SMBus-Schnittstellen der eingebauten Grafikkarte und den `i2c-tiny-usb-Adapter` (`i2c-2`, also Busnr. 2!).

Interessant wäre nun zu wissen, welche Slaves (Master ist ja der I<sup>2</sup>C-Adapter) am Bus angeschlossen sind. Auch diese Informationen liefert `i2cdetect`:

```
# i2cdetect -y 2
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- 38 -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- 49 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Man erfährt, dass zwei Slaves unter den Adressen `0x38` und `0x49` erkannt wurden. Deren Typ könnte man eventuell über den teilweise standardisierten Wert der Adresse ermitteln. In den meisten Fällen wird man aber sicherlich wissen, was angeschlossen ist. Hier sind es z.B. ein 7-Segment-Anzeige-IC `SAA1064`<sup>30</sup> (`0x38`) und

<sup>28</sup> ausführliche Informationen zu den Programmen der I<sup>2</sup>C-Tools sind den Manpages zu entnehmen

<sup>29</sup> <http://www.lm-sensors.org/wiki/I2CTools>

<sup>30</sup> <http://bralug.de/wiki/BLIT2008-Board-7-Segment-Anzeige>

ein Temperatursensor-IC LM75<sup>31</sup> (0x49).

Zum Datenaustausch mit den I<sup>2</sup>C-ICs können weitere Programme aus den I<sup>2</sup>C-Tools (`i2cget` zum Lesen; `i2cset` zum Schreiben) verwendet werden.

Auslesen der Temperaturregister (Register 0 und 1) des LM75 an Bus 2 mit der Adresse 0x49 (wobei das Ergebnis noch entsprechend zu dekodieren<sup>32</sup> ist):

```
# i2cget -y 2 0x49 0 w
0x7617
```

Initialisierung<sup>33</sup> des SAA1064 und setzen aller Segmente der ersten Anzeigestelle:

```
# i2cset -y 2 0x38 0x00 0x77
# i2cset -y 2 0x38 0x01 0xff
```

Anregungen für die C-Programmierung des I<sup>2</sup>C-Bus findet man u.a. in der Dokumentation des Kernel-Moduls `i2c-dev`<sup>34</sup>. Prinzipiell kommt man dabei mit den üblichen Funktionen zum Öffnen, Lesen, Schreiben sowie Schließen von Dateien aus, die natürlich auch in anderen Sprachen verfügbar sind.

Abschließend sei erwähnt, dass das Projekt `lm_sensor`<sup>35</sup> zahlreiche I<sup>2</sup>C-Slaves unterstützt. `lm_sensor` ist wiederum Voraussetzung für viele weitere Applikationen und Tools.

## 2.3 1-Wire vs. I<sup>2</sup>C

Wann sollte man nun welchen Bus einsetzen? Die Beantwortung dieser Frage hängt sehr stark vom jeweiligen Anwendungsszenario ab und ist teilweise auch Geschmackssache.

Klarer Vorteile des 1-Wire-Bus sind die realisierbaren Leitungslängen, die geringe Anzahl der notwendigen Drähte und die Stromversorgung der Komponenten über die Datenleitung. Dies prädestiniert diesen Bus geradezu für weitläufige Netze. Andererseits könnten die schlechtere Verfügbarkeit von unterschiedlichen Peripheriemodulen sowie diverse patentrechtliche Beschränkungen Kriterien sein, die es nötig machen nach einer anderen Lösung zu suchen.

Für den I<sup>2</sup>C-Bus spricht die hohe Anzahl der verfügbaren unterschiedlichen Baugruppen und Komponenten. Die klare Offenlegung der Standards und die eindeuti-

31 <http://bralug.de/wiki/BLIT2008-Board-Thermo>

32 siehe Datenblatt LM75

33 siehe auch Datenblatt SAA1064

34 <http://www.mjmwired.net/kernel/Documentation/i2c/dev-interface>

35 <http://www.lm-sensors.org/>

ge patentrechtliche Situation ermöglicht es ohne Probleme weitere Module zu entwickeln. So wurde z.B. vom Autor dieses Beitrages ein DCF77-I<sup>2</sup>C-Slave<sup>36</sup> mit einfachen Mitteln konzipiert und aufgebaut. Dagegen ist man durch die maximal möglichen Buslängen und den stark eingeschränkten Adressraum für die Slaves auf örtlich begrenzte Szenarien beschränkt.

Ein weiteres Auswahlkriterium könnten die erzielbaren Datenübertragungsraten sein, die bei I<sup>2</sup>C höher sind, als bei 1-Wire.

### 3 Nur zum „Teekochen“?

Bestimmt nicht! Auch dem Nicht-Teetrinker werden mit Sicherheit zahlreiche weitere Anwendungsbereiche einfallen. Überall dort, wo Sensoren Daten ermitteln und/oder Prozesse/Zustände geregelt bzw. gesteuert werden sollen, stellen die besprochenen Datenbusse lohnenswerte Betätigungsfelder dar. Der Einsatz unter Linux ist dabei kein Problem.

Das für den Betrieb der vorgestellten Bussysteme nicht immer ein PC notwendig ist, beweisen zahlreiche Projekte. Hier werden meist Mikrocontroller<sup>37</sup> als 1-Wire<sup>38</sup>- oder I<sup>2</sup>C<sup>39</sup>-Busmaster eingesetzt.

Ach so, bevor ich es vergesse, Kaffee sollte bitte nur mit 92°-96°C heißem Wasser<sup>40</sup> aufgegossen werden...!

---

36 <http://bralug.de/wiki/BLIT2008-Board-DCF77>

37 <http://de.wikipedia.org/wiki/Mikrocontroller>

38 z.B. [http://bralug.de/wiki/BLIT2008-Board\\_mit\\_1-Wire](http://bralug.de/wiki/BLIT2008-Board_mit_1-Wire)

39 z.B. <http://bralug.de/wiki/BLIT2008-Board>

40 <http://maskal.de/kaffee/zubereitung/optimal/zubereitungskriterien/wassertemperatur/>

# Entwicklung Cyber-Physikalischer Systeme am Beispiel des NAO-Roboters

**Sebastian Götz, Max Leuthäuser, Christian Piechnick, Jan Reimann,  
Sebastian Richly, Julia Schroeter, Claas Wilke und Uwe Assmann**

sebastian.goetz@acm.org, max.leuthaeuser@googlemail.com, {christian.piechnick,  
jan.reimann, sebastian.richly, julia.schroeter, claas.wilke, uwe.assmann}@tu-dresden.de  
<http://www-st.inf.tu-dresden.de/>

Eingebettete Systeme sind integraler Bestandteil unserer Umwelt und werden in zahlreichen Anwendungsbereichen eingesetzt. Obwohl ihre Vernetzung große Vorteile verspricht, arbeiten sie jedoch meist isoliert. Ein Grund dafür ist die erhöhte Komplexität ihrer Koordination und Steuerung. In diesem Zusammenhang hat sich in den letzten Jahren der Begriff der cyber-physikalischen Systeme (CPS) durchgesetzt. Dabei werden verschiedenen Systeme durch eine kausale Verknüpfung mit Cloud-basierten Softwaresystemen, unter Verwendung von abstrakten Modellen, koordiniert. Da viele eingebettete Systeme auf Linux basieren, kommt diesem Betriebssystem eine besondere Bedeutung bei der Entwicklung von CPS zu. In diesem Papier wird eine mögliche Systemarchitektur für die Integration eines Linux-basierten eingebetteten Systems in ein CPS vorgestellt. Am Beispiel eines NAO-Roboters wird gezeigt, wie Java-basierte Steuerungswerkzeuge auf Basis einer Service-orientierten Architektur eingesetzt werden können, um komplexe Kooperationsszenarien zu realisieren.

## 1 Einleitung

Eingebettete Systeme sind heutzutage in nahezu jedem Lebensbereich zu finden, sei es in Steueranlagen, Fahrzeugen oder Robotern. Traditionell arbeiten diese Systeme isoliert und Interaktionen zwischen verschiedenen Systemen sind nur für einzelne Kooperationsszenarien vorgesehen. In vielen Anwendungsszenarien sind solche Vernetzungen jedoch sinnvoll, beispielsweise für Car-2-Car-Kommunikation oder Rail Cabs [2]. In den letzten Jahren hat sich dafür der Begriff der CPS durchgesetzt. Das sind Netzwerke aus heterogenen eingebetteten Systemen, die mit einem Software-System kausal verknüpft sind. Diese neuen Anwendungsbereiche führen zu einer erhöhten Komplexität von Entwicklung und Steuerung der Geräte, die mit bestehenden Konzepten nur schwer zu adressieren ist. Um in der Lage zu sein, kooperative eingebettete Systeme wie Züge, Autos, Maschinen oder Roboter zu steuern und zu koordinieren, muss sowohl ihre Struktur, als auch ihr Verhalten modelliert werden. Zusätzlich muss ihr Zusammenspiel gesteuert werden, um die notwendigen Belange wie Energieverbrauch, Zuverlässigkeit oder Sicherheit gewährleisten zu können. Auf diese Weise werden die klassischen Softwareentwicklungsverfahren für eingebettete Systeme mit den aktuellen high-level Technologien der Modell-getriebenen Softwareentwicklung kombiniert. Viele der eingebetteten Systeme, die sich für CPS-Szenarien

eigenen, basieren auf dem Betriebssystem Linux, weshalb ihm eine besondere Bedeutung zukommt. So wird Linux beispielsweise zur Steuerung der humanoiden NAO-Roboter der Firma Aldebaran<sup>1</sup> eingesetzt. NAOs zeichnen sich durch offene und wohldefinierte Programmierschnittstellen aus, so dass sie einfach programmiert und mit anderen Systemen integriert werden können. Diese Eigenschaften machen diesen Robotertyp zum geeigneten Forschungsgegenstand für die Erprobung neuer Entwicklungsverfahren für die semi-autonome Steuerung von CPS.

In dieser Ausarbeitung werden aktuelle Forschungsfragen und -ergebnisse im Bereich der CPS am Beispiel des NAO-Roboters vorgestellt. Es wird eine mögliche Systemarchitektur für die Integration eines Linux-basierten eingebetteten Systems mit Java-basierten Steuerungswerkzeugen auf Basis einer Service-orientierten Architektur vorgestellt und gezeigt, wie diese genutzt werden kann, um komplexere Kooperationszenarien zu realisieren. Durch das eingesetzte Betriebssystem Linux ist es möglich, die Systemfunktionalität des Roboters im Zusammenspiel mit vernetzten, eingebetteten Systemen zur Verfügung zu stellen. Die weitere Ausarbeitung ist wie folgt gegliedert: In Abschnitt 2 werden Forschungsfragen im Bereich der CPS identifiziert, bevor mögliche Lösungen im Abschnitt 3 am Beispiel des NAO-Roboters vorgestellt werden. Anschließend illustriert Abschnitt 4 erzielte und zu erwartende Ergebnisse. Die Ausarbeitung schließt mit einer Zusammenfassung in Abschnitt 5.

## 2 Herausforderungen der Entwicklung cyber-physikalischer Systeme

Typische Anwendungsszenarien für CPS lassen sich in verschiedenen Branchen identifizieren. Darunter zählen Gesundheitswesen, Logistik, Verkehrswesen, Produktion und die Unterhaltungsbranche. Ein beliebtes Szenario ist der Einsatz von Robotern in der Altenpflege zur Getränke- und Medikamentenausgabe, sowie zur Unterhaltung oder für nächtliche Patrouillen. In der Logistik ermöglichen CPS die Realisierung von Just-in-time Zulieferketten und deren nahtlose Integration in die Produktion. Ein Beispiel für ein CPS im Verkehrswesen bietet die Stadt Dublin in der intelligente Ampelsteuerungen und Verkehrsführung in einer Zusammenarbeit von IBM und dem Trinity College Dublin erprobt werden [1]. Wesentliches Merkmal, das ein CPS auszeichnet, ist die enge Vernetzung mehrerer eingebetteter Systeme, die im Zusammenspiel einen autonomen Regelkreis [3] bilden. Die beteiligten Geräte übernehmen dabei verschiedene Aufgaben: *Sensoren* ermöglichen die Reflektion über einen Ausschnitt der realen Welt, *Aktuatoren* erlauben die Einflussnahme auf physikalische Prozesse. *Kommunikatoren* realisieren den Informationsaustausch zwischen den Geräten und *Rechner* stellen die notwendige Funktionalität bereit. Ein einzelnes Gerät kann dabei mehrere dieser Aufgaben bewerkstelligen. Das Gesamtsystem setzt voraus, dass alle Aufgaben umgesetzt werden. Der typische Ablauf im Regelkreis umfasst die Aufnahme von Sensorwerten (*Monitoring*), deren Abstraktion und Aggregation (*Analyse*), die Ableitung des weiteren Vorgehens basierend auf diesen Daten (*Planung*) und letztlich der Umsetzung des berechneten Plans (*Ausführung*).

---

<sup>1</sup><http://www.aldebaran-robotics.com/>

Jeder der vier Schritte birgt Probleme, wobei insbesondere Planung und Ausführung neue, für CPS spezifische Probleme, mit sich bringen. Dies ist in der Komplexität des CPS begründet, da im Gegensatz zu Einzelsystemen die Kollaboration der Geräte sowohl bei der Planung als auch bei der Ausführung berücksichtigt werden muss. Eine wesentliche Stärke von CPS liegt jedoch in der Modellbildung, die es ermöglicht, die Planung auf einer für das jeweilige System passenden Abstraktionsebene durchzuführen. Vor allem erlaubt sie damit auch, die Effizienz der Planung zu skalieren. Ein zentrales Problem der Ausführung (und indirekt auch der vorgelagerten Schritte) ist die Behandlung nicht-funktionaler Eigenschaften (NFE). Sicherheit, Energieeffizienz, Verlässlichkeit und Echtzeit-Garantien sind nur einige der zahlreichen NFE, die im Kontext von CPS eine besondere Rolle spielen. Die Integration mit der realen Welt ergibt ein Zusammenspiel Zeit-diskreter Systeme mit Zeit-kontinuierlichen Systemen, was die Notwendigkeit von Echtzeit-Garantien aufzeigt. Ebenso besteht die Möglichkeit, dass das Gerät Menschen, die Umwelt oder sich selbst gefährdet, woran die Sicherheitsanforderungen deutlich werden. Durch gegenseitige Abhängigkeiten zwischen NFE ergeben sich weitere unabdingbare Anforderungen. Zum Beispiel kann in elektrisch betriebenen Kraftfahrzeugen eine leere Batterie bei voller Fahrt sowohl das Kraftfahrzeug, die Insassen, als auch die Umgebung gefährden.

Bei der Adressierung dieser Probleme spielt die Softwaretechnologie eine wesentliche Rolle. Abgesehen von Monitoring- und Analyseverfahren, die auch für einfache eingebettete Systeme benötigt werden, stellen die Planung und Ausführung neue Herausforderungen an die Softwaretechnologie. Diese sind vor allem der Einsatz von Modellen zur Laufzeit für eine skalierende Planung und neue Adaptationstechniken, die den kollaborativen Charakter von CPS unterstützen. Zusätzlich verlangt das Ziel der effizienten Nutzung von Energie neue Techniken der Softwaretechnologie. Das Testen von Energieaspekten bei der Entwicklung von Software, Energie-Monitoring und Energie-optimierende Planungstechniken sind einige dieser Herausforderungen. Nicht zuletzt stellt die Integration der Einzelgeräte ein zentrales Problem von CPS dar. Dies liegt vor allem in der Heterogenität der Geräte begründet. Eine Lösung stellen einheitliche Schnittstellen zwischen den kollaborierenden Geräten dar. Um eine kooperative Überwachung, Koordination und Steuerung für unterschiedliche Plattformen, Programmiersprachen und Datenformate gewährleisten zu können, eignen sich Webservices hervorragend. Diese können beispielsweise mittels BPEL zu komplexen Prozessen komponiert werden, wodurch die Komplexität bei der Koordination verschiedener heterogener Systeme durch ein abstrakteres Modellierungswerkzeug reduziert wird. Darüberhinaus können in der Software-Cloud gehostet werden, um typische Middleware-Anforderungen wie Orts-, Sprach- und Plattformtransparenz, als auch Skalierbarkeit gewährleisten zu können.

### **3 NAOs als Beispiel für Devices as a Service**

Wie im vorangegangenen Abschnitt erläutert, stellt die Integration von Einzelgeräten in ein CPS ein zentrales Problem für deren Entwicklung dar. In diesem Abschnitt wird



Abbildung 1: Geschichtete Architektur des NaoService [5].

beispielhaft gezeigt, wie dieses Problem über einen einfachen *Device as a Service*-Ansatz mittels Webservice gelöst werden kann. Als Anwendungsbeispiel wird dafür die Schnittstelle eines NAO-Roboters als Webservice für die Integration in verteilte Anwendungen bereitgestellt. NAO-Roboter werden mit einem Linux-Betriebssystem inklusive Python ausgeliefert. Der Webservice wurde daher in Python implementiert, wobei das Web-Framework Bottle<sup>2</sup> die Grundlage bildet und dabei sehr ressourcensparend arbeitet. Der Webservice wird beim Start des Betriebssystems auf den NAOs initialisiert. In der Literatur findet sich häufig der Vorschlag, serviceorientierte Architekturen mit Hilfe von SOAP (simple object access protocol) oder CORBA (common object request broker architecture) zu realisieren [4, 6, 7, 8, 10]. Beide Techniken nutzen dafür eine eigenes Datenformat. Dies ist problematisch da das Verarbeiten dieser Datenpakete die begrenzten Ressourcen des NAOs stärker beansprucht und damit unnötigen Kommunikationsoverhead verursachen. Deshalb wurde der sogenannte *NaoService*<sup>3</sup> auf Basis von REST (Representational State Transfer) umgesetzt. REST kommuniziert über HTTP und adressiert mittels URIs.

Die dem NaoService zugrunde liegende Architektur besteht aus mehreren Kommunikationsschichten, die sich zu zwei Blöcken zusammenfassen lassen (vgl. Abb. 1). Der untere Block beinhaltet drei Schichten, welche REST implementieren und auf dem NAO bereitstellen. Der obere Block stellt dagegen die Anbindung und Nutzung der bereitgestellten Webservices auf einem anderen Gerät dar, welches die REST-Schnittstelle verwendet um einen oder mehrere Roboter kontrollieren zu können (Koordinator). Die eigentliche Schnittstelle für die Steuerung der NAOs ist NaoQi, welches die unterste und hardwarenächste Schicht der Architektur darstellt. NaoQi bietet eine eigene API um die Aktoren und Sensoren eines individuellen NAOs anzusprechen. Es ist selbst in C++ implementiert und wird von Aldebaran bereitgestellt. Das Unternehmen bietet ebenfalls eine entsprechende Python-API an (Python-Brücke), welche die Fähigkeiten von NaoQi für Python verfügbar macht. Darauf

<sup>2</sup><http://www.bottlepy.org/>

<sup>3</sup><http://code.google.com/p/naoservice/>

aufbauend findet sich der NaoService, welcher diese API als REST-Service für die Außenwelt zur Verfügung stellt. Diese Webservice-Schicht wird größtenteils automatisch aus der Python-Brücke mithilfe der Introspection-Fähigkeiten Pythons generiert. Dieser Umstand verringert nicht nur die Entwicklungszeit des Webservices enorm, es erleichtert auch die Anpassung der Schnittstelle, sollte sich NaoQi in einer zukünftigen Version ändern. So werden nur die Schnittstellen bereits gestellt, die auf dem jeweiligen NAO gerade verfügbar sind. Neben der Tatsache, dass der Webservice direkt über HTTP angesprochen werden kann, werden auch verschiedene HTML-Formulare erzeugt, welche den Zugriff auf die API via Webbrowser ermöglichen (vgl. Abb. 2(a)). Mit Hilfe eines Codegenerators werden automatisiert Java-Proxies erzeugt, welche den Zugriff auf den Webservice kapseln. Diese Proxies erlauben es, auf alle Module mit ihren Methoden, wie sie durch die API NaoQis angeboten werden, zuzugreifen. Die Generierung erfolgt automatisch aus den C++-Header-Dateien NaoQis. Die Aufrufe an den Webservice sind entsprechend generisch daraus abgeleitet. Der Vorteil dieses Ansatzes der Generierung liegt darin, dass derartige Proxies für beliebige andere Programmiersprachen erzeugt werden können. Damit ist es möglich, andere Programmiersprachen nach dem Plug-and-Play-Prinzip für die Steuerung des NAOs vorzubereiten und mit dem NaoService zu verbinden. Für die Ansteuerung der NAOs aus Java wurden über den Proxy-Klassen noch manuell Utility-Methoden erstellt, die die Kommunikation aus Java weiter erleichtern. Darüber liegt als letzte Schicht die eigentliche Anwendung, welche nun in der Lage ist, NAOs in einem kollaborativen Szenario zu steuern.

Zusammengefasst bietet der vorgestellte Service-orientierte Ansatz eine einfache Möglichkeit zur Kommunikation mit einzelnen oder mehreren (NAO-)Robotern. Er kann einfach für andere Roboterarchitekturen angepasst oder erweitert werden. Der untere Block mit seinen drei Schichten (NaoQi, Python-Brücke, NaoService) kann zudem lokal auf jedem Rechner ausgeführt werden und erlaubt damit die Simulation eines NAOs und eine NAO-in-the-Loop-Software-Entwicklung.

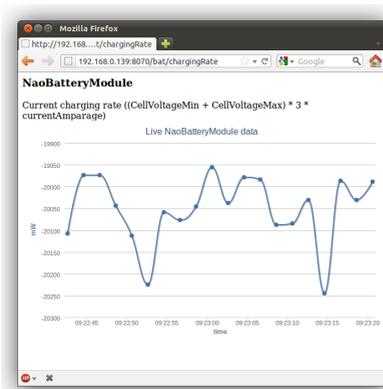
## 4 Ergebnisse

Die im vorangegangenen Abschnitt vorgestellte Service-Architektur für NAOs wurde im Forschungsprojekt QualiTune<sup>4</sup> erfolgreich umgesetzt. Darauf basierend wurde der NAO-Roboter in verschiedene Anwendungsszenarien eingebunden, welche die Anwendbarkeit des *Device as a Service*-Ansatzes belegen.

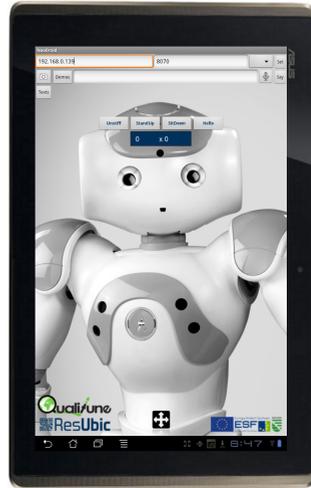
**NaoDroid** Mit der beschriebenen Integration des NaoServices in Java ist es nun möglich, Java-Anwendungen zu entwickeln, die über mit NAO-Robotern kommunizieren. So wurde beispielsweise eine Java-basierte Steuerungssoftware namens *NaoDroid* implementiert, die auf Android-basierten Smart Phones und Tablet PCs genutzt werden, um NAO-Roboter fernsteuern zu können. Abb. 2(b) zeigt einen Screenshot der NaoDroid App.

---

<sup>4</sup><http://www.qualitune.org/>



(a) Web-Schnittstelle des NaoService.



(b) NaoDroid App auf einem Android Tablet.

Abbildung 2: Nao-Steuerung über NaoService (a) und NaoDroid (b).

**NaoText** Kollaborative Szenarien für CPS und Roboter sind eine Herausforderung, da entsprechende Abstraktions- und Beschreibungsmöglichkeiten des kollaborativen Verhaltens fehlen. Aus diesem Grund wird derzeit im Projekt QualiTune eine neuartige Programmiersprache zu entwickelt, die es ermöglicht, NAOs kollaborativ zu steuern. Eine erste Version dieser Sprache wurde in [5] vorgestellt. Derzeit wird ein Scalabasierter Interpreter für NaoText entwickelt, welcher dessen rollen-basierten Sprachkonzepte von NaoText unterstützen soll.

**JouleUnit** Ebenfalls basierend auf dem NAO-Service ist eine Erweiterung des JUnit-Frameworks entstanden, die es erlaubt, über Java verschiedene Anwendungen auf NAOs auszuführen und dabei ihren Energieverbrauch zu erfassen. So erlaubt es JouleUnit [9]<sup>5</sup>, Energietestfälle zu definieren, die den maximalen Energieverbrauch einer Anwendung überwachen, oder per Regressionstest helfen, Anwendungen bezüglich ihres Energieverbrauchs zu optimieren.

**Aktuelle Entwicklungen** Die zuvor genannten Beispiele belegen bereits, dass sich der NaoService hervorragend eignet, um basierend darauf zahlreiche verschiedene Anwendungen und Anwendungsszenarien zu realisieren. Derzeit wird im Projekt QualiTune im Rahmen einer studentischen Arbeit ein Ansatz entwickelt, der darauf abzielt, NAO-Roboter über Petri-Netze zu modellieren und zu steuern. Die Petri-Netze beschreiben dabei das mögliche Verhalten des NAOs (Zustand der einzelnen Motoren und mögliche Zustandsübergänge) und verwenden zur Ansteuerung des NAOs

<sup>5</sup><http://www.jouleunit.org/>

ebenfalls den NaoService. Eine weitere studentische Arbeit versucht, ein Verschleißmodell für NAOs zu entwickeln. Mithilfe von neuronalen Netzen wird die Hitzeentwicklung der im NAO verbauten Motoren über die Zeit in Abhängigkeit von ihrer Ansteuerung beschrieben. Damit lässt sich vorhersagen, wie lange einzelne Motoren noch sinnvoll genutzt werden können, bevor sich diese wegen Überhitzung selbstständig abschalten und so den NAO-Betrieb einschränken.

## 5 Zusammenfassung

Die zunehmende und umfassende Einbeziehung heterogener eingebetteter Systeme in den Alltag erfordert eine stärkere Integration und Kopplung der einzelnen Teilsysteme. Herkömmliche low-level Technologien bieten einen zu niedrigen Abstraktionsgrad, um die Komplexität beherrschbar zu machen, die bei der Planung kooperativer Aufgaben für heterogene Systeme in einer offenen Welt entsteht. Aus diesem Grund müssen Modellierungsverfahren eingesetzt werden, die Repräsentationen der physikalischen Ressourcen und Systeme nutzen, um systemübergreifende Analyse- und Planungsoperationen durchführen zu können. Um eine sprach- und plattformunabhängige Koordination zu realisieren, bieten sich Webservices als Kommunikations- und Zugriffsschicht an, da neben systemunabhängigen Funktionsaufrufen und der breiten Unterstützung in verbreiteten Programmiersprachen auch eine einfache Integration in die Software-Cloud möglich ist.

Der NAO-Roboter ist ein Linux-basierter humanoider Roboter, der über eine offene Schnittstelle mit anderen Softwaresystemen integriert werden kann. Aus diesem Grund eignet er sich sehr gut, um die theoretischen Konzepte hinsichtlich ihrer Praxistauglichkeit zu überprüfen. Dafür wurde ein reflektiver Webservice entwickelt, der die bestehende Programmierschnittstelle über die Systemgrenzen hinweg verfügbar macht, um das *Monitoring* und die *Steuerung* des Roboters an zentraler Stelle realisieren zu können. In diesem Rahmen wurden verschiedene Werkzeuge entwickelt, die unterschiedliche Interaktionskonzepte zur Steuerung von Robotern aufzeigen und zur aufbereiteten Sammlung von Sensormesswerten dienen. In zukünftigen Arbeiten werden diese Werkzeuge mit dynamischen Prozessmodellen kombiniert, um eine semi-automatische Steuerung in einer potentiell offenen Welt zu verwirklichen. Darüber hinaus verspricht die Anwendung weiterer softwaretechnologischer Konzepte, wie die dynamische Adaption der Anwendungsarchitektur durch Optimierung nicht-funktionaler Eigenschaften, große Vorteile bei der Minimierung des Energieverbrauchs oder der Gewährleistung von Sicherheitsanforderungen.

## Danksagung

Diese Arbeit wurde vom Europäischen Sozialfond (ESF) und dem Freistaat Sachsen im Forschungsprojekt ZESSY #080951806, von der Deutschen Forschungsgemeinschaft im SFB 912 (HEAC) und vom ESF und der SAP AG im Forschungsprojekt #080949335 gefördert.

## Literatur

- [1] Future Cities Projects at Distributed Systems Group of Trinity College Dublin. <http://www.dsg.scss.tcd.ie/FutureCities>, January 2012.
- [2] RailCab-Projekt: Forschungsinitiative Neue Bahntechnik Paderborn. <http://www.railcab.de/>, January 2012.
- [3] S. Dobson, S. Denazis, A. Fernández, D. Gaiti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):223–259, 2006.
- [4] L. Flückiger, V. To, and H. Utz. Service-Oriented Robotic Architecture Supporting a Lunar Analog Test. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (ISAIRAS)*, 2008.
- [5] S. Götz, M. Leuthäuser, J. Reimann, J. Schroeter, C. Wende, C. Wilke, and U. Aßmann. A Role-based Language for Collaborative Robot Applications. In *1st International ISoLA Workshop on Software Aspects of Robotic Systems, ISOLA SARS'11*, 2011.
- [6] Y.-G. Ha, J.-C. Sohn, and Y.-J. Cho. Service-Oriented Integration of Networked Robots with Ubiquitous Sensors and Devices using the Semantic Web Services Technology. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS2005)*, pages 3947–3952, 2005.
- [7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.
- [8] V. M. Trifa, C. M. Cianci, and D. Guinard. Dynamic Control of a Robotic Swarm using a Service-Oriented Architecture. In *13th International Symposium on Artificial Life and Robotics (AROB2008)*, 2008.
- [9] C. Wilke, S. Götz, J. Reimann, and U. Aßmann. Vision Paper: Towards Model-Based Energy Testing. In *Model Driven Engineering Languages and Systems (MODELS 2011)*, volume 6981 of LNCS, pages 480–489. Springer, 2011.
- [10] B. Wu, B.-H. Zhou, and L.-F. Xi. Remote multi-robot monitoring and control system based on MMS and web services. In *Industrial Robot: An International Journal*, volume 34, pages 225–239. 2007.

# Gründen mit freier Software

**Georg Schütz – Dipl.Wirtschaftsingenieur (FH)**  
KaMUX GmbH & Co. KG – <http://kamux.de>  
[georg.schuetz@kamux.de](mailto:georg.schuetz@kamux.de)

## 1 Vorwort

In Deutschland stellen Kleine und Mittlere Unternehmen (KMU) etwa 95% aller Betriebe. Im Jahr 2010 wurden rund 450.000 neue Unternehmen in dieser Größenklasse gegründet (Quelle: Statistisches Bundesamt). Mittlerweile arbeite ich seit 9 Jahren (nicht nur) als Gründungsberater. Meine Erfahrung zeigt, dass freie Software noch immer eine eher unbekannte Größe ist. Die Zahl der Gründungen und der mangelnde Bekanntheitsgrad freier Software sind Grund genug, sich mit der Problematik auseinanderzusetzen.

Gründer haben es schwer, IT-Dienstleister auch. Man muss an viele Dinge denken, sich organisieren, sich finanzieren, auf dünner Decke tragende Säulen errichten, mit wenig Geld auskommen und sein Geschäft zum Blühen bringen. Freie Software kann hier in vielerlei Hinsicht ein wichtiger Baustein sein. Einerseits für den Nicht-IT-Unternehmer, weil er professionelle Werkzeuge nutzen kann, ohne Tausende zu investieren und nur wirklichen Nutzen bezahlt. Und natürlich für den IT-Unternehmer, weil er seinen Kunden diese Werkzeuge näher bringen kann und nicht an Lizenzen sondern am Nutzen und der Dienstleistung verdient, die seine Kunden abnehmen. Gerade das ist ein Aspekt, der IT-Dienstleister in die Lage versetzt, sich von einem großen Teil der Konkurrenten abzuheben.

## 2 Wie gründet man mit IT?

Die Frage ist durchaus zweideutig gemeint. Der „normale“ Gründer hat heute kaum die Möglichkeit, ohne IT zu gründen. Alleine die gesetzlichen Anforderungen an die Übermittlung von steuerlichen Sachverhalten erfordern es, dass dem Unternehmer funktionsfähige und sichere IT zur Verfügung steht. Obwohl es immer noch Gewerbe gibt, in denen es vorstellbar ist, keine IT zu nutzen, gehört es mittlerweile

zur Normalität eine Mailadresse zu haben und darüber auch ansprechbar zu sein. Der „IT-Gründer“, also ein Dienstleister bzw. Neuunternehmer im IT-Bereich gründet sowieso mit IT-Unterstützung. Alles andere wäre wohl verwunderlich.

Mit dem folgend gebrauchten Begriff „Infrastruktur“ meine ich nicht nur die physikalischen Gegebenheiten sondern mehr noch die inhaltlich-organisatorischen Funktionen, die ein Unternehmen benötigt. In der Praxis lässt sich das Phänomen beobachten, dass sowohl der „normale Gründer“ als auch der „IT-Gründer“ wenig Wert auf die eigene IT-Infrastruktur-Ausstattung legen und eine konzeptionelle Vorbereitung der eigenen IT/Organisation für das Geschäft nicht stattfindet. Bei einem nicht-IT-Unternehmer ist das verständlich, dessen Hauptaugenmerk liegt auf seinem Gewerbe, seinen Produkten oder Leistungen. Die IT-Gründer verhalten sich dabei aber genau so! In den meisten Fällen liegt auch deren Fokus auf dem Vertrieb und Verkauf und am wenigsten auf der eigenen Organisation und Infrastruktur.

Interne Organisation, Regeln und Abläufe werden anscheinend von den meisten Gründern als entbehrlich für den ersten Schritt erachtet. Die eigene Organisation kritisch zu betrachten und gleich von Anfang an eine zukunftssichere Lösung einzuführen, empfinden viele Gründer als zu kompliziert, „wir wollen jetzt erst mal so starten“ lautet die Devise. Klein Gründungen erfolgen oft mit einer einzigen Hardware, z.B. einem Notebook, weil da „ja alles ab Werk drauf ist“ und die Organisation passt sich dann dem Arbeitsmittel an. Als Adressverwaltung dient der Mailclient, vertriebliche Prozesse werden mit einer Kalkulationstabelle abgebildet und Angebote, Rechnungen usw. mit der Textverarbeitung geschrieben. Die Themen Datenschutz, Datensicherheit, Effizienz der Datenhaltung (inkonsistente Redundanz) oder späteres Wachstum mit mehreren beteiligten Personen spielen zu diesem Zeitpunkt keine Rolle. Software zur Unternehmenssteuerung wird als zu teuer, zu umfangreich, zu schwierig zu verstehen oder schlicht als „für mich unnötig“ betrachtet.

### **3 Was hindert Gründer (und bestehende Unternehmen) am Geldverdienen?**

Aus meinen Beobachtungen in den letzten 9 Jahren zeigen sich im wesentlichen die folgenden Gründe dafür, dass die Möglichkeiten des Geldverdienens nicht oder nicht effizient ausgeschöpft werden können:

1. die verpflichtenden „unproduktiven“ Zeiten nehmen zu

Auch wenn die Politik permanent den Bürokratieabbau predigt, so ist gerade für Kleine- und Kleinst-Unternehmen ein Anstieg der Bürokratiezeiten festzustellen. Die Verpflichtungen werden penibler, die Meldezyklen wer-

den kürzer, z.B. bei der innergemeinschaftlichen „Zusammenfassenden Meldung“ oder der verpflichtenden monatlichen Umsatzsteuer-Voranmeldung (USt-VA), die beide früher vierteljährlich abgegeben werden mussten.

2. die vorhandenen Hilfsmittel unterstützen die Prozesse nicht

Wie oben beschrieben wird bei der Gründung die Organisation „irgendwie momentan zweckmäßig“ begonnen. Mit steigendem Anfrage- oder Auftragsvolumen wird es immer schwerer bis unmöglich, den Kunden Rechnung zu tragen. Die Folge sind zu hohe Bearbeitungszeiten der einzelnen Anfragen, permanente Arbeitsunterbrechungen durch parallel laufende gleichartige Prozesse oder Rückfragen.

An sich notwendige Aktionen, wie z.B. das telefonische Nachverfolgen von Angeboten werden mangels verfügbarer Zeit nicht durchgeführt. An sich gute Angebote führen dann nicht zu Aufträgen.

3. die Gründungsprozesse funktionieren im Wachstum nicht mehr

Kommt es (hoffentlich) zum Wachstum des Unternehmens, dann sind schnell mehrere Mitarbeiter im Spiel. Plötzlich wird das Thema „wie greifen alle auf die gleichen Daten zu“ wichtig. Oft wird diese Problematik durch eine zweite Ein-Hardware-Lösung angegangen, es stellt sich dann die Frage der Synchronisierung.

Dadurch, dass die Prozesse ursprünglich auf den Ein-Personen-Betrieb ausgelegt waren, hat der erfolgreiche Gründer jetzt eine doppelte Herausforderung zu bewältigen: die Neuorganisation der Arbeitsabläufe und den betrieblichen Erfolg. Während er mehr oder größere Aufträge abarbeitet soll oder muss er sich Gedanken um die inneren Strukturen machen Suboptimale Entscheidungen sind so vorprogrammiert.

4. Sicherheit und Verfügbarkeit der Systeme

Kommt es bei der oben beschriebenen Ein-Hardware-Lösung zu einem Problem, dann steht das ganze Unternehmen. Eine an sich triviale Erkenntnis, die leider in der Praxis viel zu oft ausgeblendet wird. Trotz entsprechend zahlreicher Meldungen und Nachrichten sind Diebstahlprävention, Datenschutz, Viren- Spam und Spywareschutz, Datensicherung (und Restore-Test!) noch immer völlig vernachlässigte Themen.

5. neue Hilfsmittel und Möglichkeiten werden unzureichend genutzt

Auch hier schlägt sich der Mangel an Zeit nieder. Um neue Hilfsmittel einzusetzen, muss man sich damit beschäftigen können und diese gegebenenfalls für die eigene Organisation anpassen oder nutzbar machen.

Ein aktuelles Beispiel: das Steuervereinfachungsgesetz 2011 erlaubt es, Rechnungen auch digital als pdf-Dokument zu versenden oder zu erhalten. Dazu benötigt das Unternehmen dann zwingend ein elektronisches Archiv zur Ablage. Der nutzbringende Einsatz des eArchivs wird aber nur dann möglich, wenn er an die Unternehmensabläufe angepasst wird. Verbringt man zu viel Zeit im operativen Geschäft (Punkt 2, 3) oder bei wachsender Bürokratie (Punkt 1) oder mit der Sicherung der Verfügbarkeit (Punkt 4), dann wird man keine Zeit dafür erübrigen können. Die an sich sinnvolle Nutzung eines eArchivs wird dann zum weiteren Hemmschuh (Punkt 2). Analog zu diesem Beispiel lassen sich viele andere Prozesse oder Werkzeuge nennen, die mit ihrer unkontrollierten oder erzwungenen Einführung die Lage „verschlimmbessern“.

## **4 Gründen mit freier Software (alle Gründer)**

Die konsequente Nutzung freier Software schon ab der Gründung bietet für den Unternehmensgründer direkte Vorteile. Vordergründig wird oft zuerst die Kosten- seite betrachtet und auf die entfallenden Lizenzgebühren verwiesen. Dieser Punkt ist bei den oben skizzierten Ein-Hardware-Lösungen in der Regel nicht stichhaltig. Die gekaufte IT-Ausstattung enthält oft sämtliche für den Bürobetrieb notwendigen Programme im sogenannten „Bundle“. Betriebssystem und Software sind also mit eingepreist. Der Kostenvorteil wird in diesem Fall erst dann signifikant, wenn der zweite Arbeitsplatz ausgestattet werden soll.

Viel wichtiger ist meiner Ansicht, dass mit dem Einsatz freier Software sichere, wachstumsfähige und stabile Prozesse von Anfang an möglich gemacht werden. Es gibt etliche F/OSS-Projekte, die kleinen Unternehmen mehr oder minder vollständige IT-Infrastrukturen zur Verfügung stellen, stellvertretend seien hier Zentyal (<http://www.zentyal.org>), der invis-Server (<http://www.invis-server.org>) und das KMUX-Projekt (<http://kmux.sourceforge.net>) genannt. Der Unterschied dieser freien Projekte liegt in der Integration der angebotenen Anwendungen bzw. Serverlösungen.

Gründer und bestehende KMU (Kleine und Mittlere Unternehmen) übersehen gerne, dass für sie als Unternehmen die gleichen Regeln gelten wie für die wenigen Großunternehmen in Deutschland auch. Dort sind aber ganze Abteilungen und Stäbe mit Organisation/IT/Sicherheit/Recht etc. befasst, Dinge die im KMU der Unternehmer in der Regel selbst bewältigen muss. Freie Software löst die auftretenden Fragen nicht per se, aber mit dem gezielten und kombinierten Einsatz von freier Software und offenen Standards lassen sich integrierte Prozesse viel einfacher gestalten.

Als Beispiel soll das KMUX-Projekt dienen. Bei der Installation eines individuellen KMUX-Systems könnten sich IT-Dienstleister und Gründer überlegen, folgende Applikationen einzusetzen:

1. SugarCRM zur Adressverwaltung und für die vertriebliche Arbeit, eventuell mit Erweiterung zum Schreiben von Angeboten und Rechnungen
2. eArchiv zur sicheren Ablage erzeugter und eingehender Rechnungen
3. Groupware-Lösung für Mail, Termine und Aufgaben
4. Terminalserver für die Arbeit direkt am Server (Server = Server + Arbeitsplatz in einem Gerät) oder mit ThinClients
5. Mailserver

Die anderen möglichen Anwendungen wie ERP, Workflow-, Dokumentationssystem oder CMS wurden bei der Auswahl für den konkreten Unternehmer ausgeschlossen.

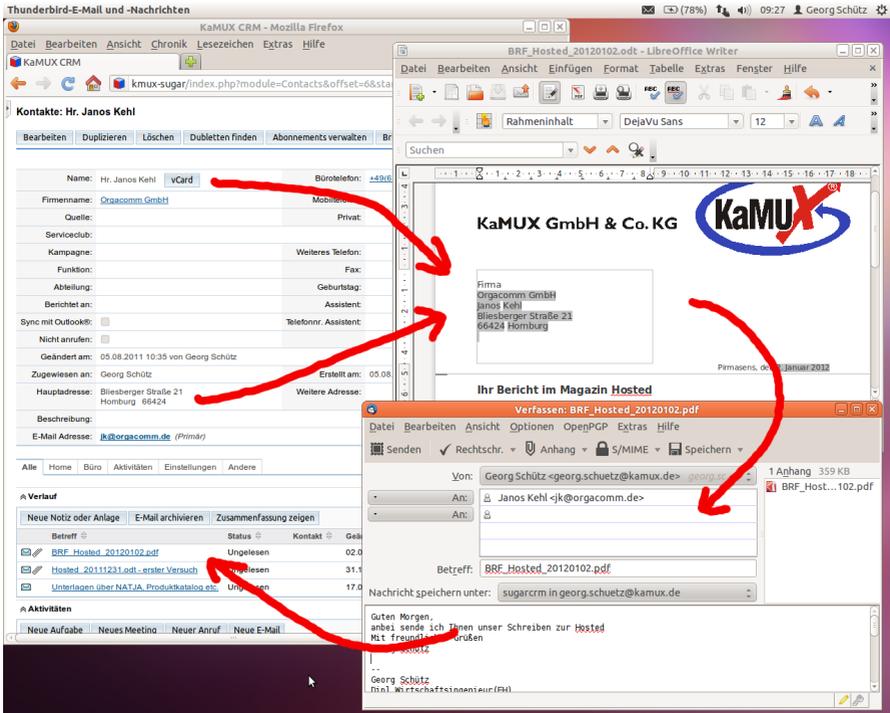
Die Installation der ausgesuchten Anwendungen in einer virtualisierten Umgebung an sich ist mit dem KMUX-Projekt nicht schwierig. Sinnvoll wird diese Konfiguration aber erst, wenn die Applikationen „miteinander sprechen“. Ein Beispiel:

Der Benutzer, der am System angelegt wird, muss automatisch vom System seine Firmen-Mailadresse, Zugriffsrechte für alle Unternehmensanwendungen und Dateiablagen bekommen, sonst wird die Systemverwaltung zu kompliziert für einen Nicht-IT-Unternehmer. Die Benutzerverwaltung muss also mit den Anwendungen kommunizieren, was der Benutzer darf und was nicht.

Die Ablauf, den ich jetzt beschreibe skizziert die darauf folgende Abbildung:

Die Kunden- und Lieferantenadressen aus SugarCRM müssen sich sofort und ohne Synchronisation in der Groupware (z.B. die Mailadresse) nutzen und möglichst ebenso direkt auf dem Terminalserver in einer Office-Anwendung wie LibreOffice verwenden lassen.

Der in LibreOffice mit den Adressen aus SugarCRM verfasste Brief an einen Kunden oder Lieferanten muss direkt per Mail/Groupware verschickt werden können und beim Versand wieder den Weg in die Historie von SugarCRM finden, damit dort alle Aktivitäten vollständig erfasst werden. Die in SugarCRM oder LibreOffice erzeugte Rechnung muss als pdf verschickt werden können und danach gegebenenfalls per Dateiablage oder Druck in das elektronische Archiv gespeichert werden.



Die Liste der durch Anwendungskommunikation sinnvoll zu gestaltenden Abläufe lässt sich nahezu beliebig verlängern. Wichtig ist, dass die Abläufe von Unternehmen zu Unternehmen unterschiedlich sind und sein können. Die Zusammenarbeit freier Softwareprodukte über offene Schnittstellen und Standards zusammen mit einer sinnvollen Vorkonfiguration ermöglicht und erleichtert die betriebliche Individualität. Gerade das ist aber eine der herausragenden Eigenschaften kleiner Unternehmen: die individuelle und flexible Befriedigung der Kundenwünsche.

Wichtig: die hier beispielhaft beschriebenen Arbeitsweisen sind unabhängig von der Zahl der beteiligten Personen im Unternehmen! Die oben geschilderte Wachstumsproblematik stellt sich nicht mehr, die Punkte 2, 3 und 4 aus dem Abschnitt 3 *Was hindert Gründer (und bestehende Unternehmen) am Geldverdienen?* sind von Anfang an erledigt.

Wie geht man in diesem Zusammenhang mit proprietärer Branchensoftware um, die es z.B. nur für Microsoft-Betriebssysteme gibt? Hier gilt es zu überlegen, wie man diese an sich geschlossenen System zumindest so weit integrieren kann, wie es sinnvoll ist. Einige geschlossene Lösungen bieten mittlerweile Schnittstellen zu Open/LibreOffice an, z.B. Lexware oder Hamburger Software. In anderen Programmen ließe sich z.B. eine Kombination mit dem eArchiv denken und herstellen. An

dieser Stelle muss man sich das konkrete Vorhaben ansehen. Auf den Punkt gebracht: die sinnvolle Kombination freier und proprietärer Software kann für das gründende Unternehmen die sinnvollste Lösung sein.

## 5 Gründen mit freier Software (IT-Dienstleister)

Wenn man sich als IT-Dienstleister selbstständig macht und die bis hierher gemachten Ausführungen als richtig empfindet, dann bietet das strukturierte Gründen mit freier Software neben der besseren eigenen Organisation weitere Vorteile:

1. Wenn ich die oben genannten Lösungen selbst einsetze, kann ich sie guten Gewissens auch meinen Kunden anbieten. Der IT-Lieferant wird zum Lösungslieferanten.
2. Wenn ich die oben genannten Lösungen selbst einsetze, kann ich sie für meine eigenen Zwecke anpassen oder um zusätzliche Bausteine ergänzen. Beispiel: eine Abrechnungssoftware gekoppelt an SugarCRM. Diese Lösung kann ich dann wieder meinen Kunden andienen, die eine ähnliche Problematik aufweisen wie mein eigenes Geschäft.
3. Wenn ich die oben genannten Lösungen selbst einsetze, dann kann ich dem Kunden erklären, dass er dafür keine Lizenzkosten hat. Das so sinnvoll gesparte Geld, kann er dann in für ihn nützliche Dienstleistungen bei mir investieren. Für ihn ein Nullsummenspiel, für mich als Dienstleister viel besser als eine Lizenz zu verkaufen.
4. Wenn ich die oben genannten Lösungen selbst einsetze, dann kann ich mich in die freien Projekte einbringen und für mich und meine Kunden neue „Produkte“ mit erarbeiten. Eine zusätzliche, individuelle Verkaufschance, die proprietäre Anbieter in der Regel nicht haben.
5. Wenn ich konsequent freie und open source Software auch für meinen Kunden einsetze, dann kann ich gegenüber der Konkurrenz auch bei den Kosten punkten.

Ein Beispiel aus einem aktuellen Projekt (Herbst 2011), bei dem ich die IT-Konzeption erstellt habe: eine geforderte Clusterlösung (2 Knoten) auf Basis VMWare VSphere kostete beim teuersten Anbieter rund 29.000 EUR netto, die inhaltlich gleiche Lösung mit Proxmox/KVM realisiert kostete bei gleicher Dienstleistungshöhe noch 19.000 EUR netto. Die Entscheidung hat der Kunde schnell getroffen. Die Ersparnis war signifikant, die Stabilität beider Lösungen ließ sich in einer Demo-Installation leicht nachweisen.

6. Vereinfachung durch Vereinheitlichung – wenn ich als IT-Dienstleister von Anfang an konsequent auf die gleichen Produkte, Installationsszenarien und einheitliche interne Abläufe setze, dann spare ich viel Zeit und Dokumentationsaufwand weil die sich die wesentlichen Dinge nicht ändern.

## 6 Fazit

Freie Software für den Unternehmenseinsatz wird noch zu selten auch in öffentlichen Einrichtungen und bei Anlaufstellen wie IHK, HWK und Gründungsberatungen propagiert und bekannt gemacht. Durch die Hard- und Softwarebundles der Hersteller ist eine grundlegende Öffnung und Information von Gründern und Verbrauchern auch nicht in Sicht. Es sollte daher ein Ziel gerade von Veranstaltungen wie den Chemnitzer Linux-Tagen und auch von IT-Dienstleistern sein, die Verbreitung freier Software in Unternehmen zu fördern. Über den „Umweg“ von der täglichen Arbeit an den heimischen Rechner wird so indirekt auch die private Nutzung freier Software voran gebracht.

Freie Software ist kein Allheilmittel. Es gibt viele proprietäre Branchenlösungen, die für Gründer und bestehende Unternehmen sinnvoll sind, und für die es (noch) keine adäquaten Pendanten in der freien Softwarewelt gibt. Daher muss man sich fragen, wie man die besten Kombinationen aus proprietärer und freier Software für den Unternehmenseinsatz findet.

Freie Software kommt für Gründer immer dann besonders in Frage, wenn sie keine spezifischen Branchenanforderungen an die IT haben oder wenn es um die grundlegende Infrastruktur für Büro- und oder Netzwerkbetrieb geht.

Die für den betrieblichen Einzelfall gezielt gewählte Kombination aus freier und proprietärer Software wird in vielen Fällen den Anforderungen der Gründer und Unternehmen das Optimum für ihren Anwendungsfall bieten können.

# Lilypond

## Ein wenig Revolution muß sein

David Kastrup  
dak@gnu.org

Im letzten halben Jahr ist erhebliche Bewegung in die Entwicklung des Notensatzprogramms Lilypond gekommen. Der Vortragende selbst ist verantwortlich für beträchtliche Erweiterung, Systematisierung und Vereinfachung der Eingabesyntax sowie einem deutlich gesteigerten Leistungsumfang von im Schemedialekt Guile geschriebenen Erweiterungen. Der Vortrag beschreibt die aktuellen Entwicklungen und Programmiermodelle.

### 1 Präambel

Sofern in diesem Dokument Experimente mit LilyPond durchgeführt werden, ist ohne besonderen Hinweis davon auszugehen, daß es sich hierbei um die zum Zeitpunkt der Drucklegung noch nicht existente stabile Version 2.16 handelt. Sollte diese zum Zeitpunkt des Lesens noch nicht vorliegen, oder sollte hier eingesetzte Funktionalität trotz größter Anstrengungen des Autors nicht mehr in diese Version aufgenommen worden sein, ist das neueste „Entwicklerrelease“ zu nutzen. Es handelt sich jedenfalls nicht um Vaporware: alle Bestandteile sind lauffähig und mindestens im Stadium der öffentlichen Codebeschau.

### 2 Zur Einordnung von LilyPond

LilyPond ist ein Notensatzprogramm. Ähnlich wie bei  $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  (<http://www.dante.de/tex.html>) als Textsatzprogramm ist es schwierig zu definieren, in welcher Konkurrenz es zu kommerziellen Produkten steht, weil kommerzielle Produkte wesentlich durch ihre Benutzeroberfläche geprägt sind.

LilyPond hat keine Benutzeroberfläche, die sich zwischen die Funktionalität des Programms und den Benutzer drängen und ihm eine Arbeitsweise vorgeben würde.<sup>1</sup>

---

<sup>1</sup>Ähnlich wie bei  $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  gibt es Unterstützung bei der Erstellung des Quelltexts durch den Anwender einerseits durch Zusatzmodi altherwürdiger Editoren wie Emacs (<http://www.gnu.org/software/emacs>) und Vim (<http://www.vim.org>) und jEdit (<http://jedit.org>). Des weiteren gibt es Spezialeditoren wie Frescobaldi (<http://frescobaldi.org/>), die dem Anwender die volle Kontrolle über beliebige Quelltexte lassen, ihn aber nach KrÄften unterstützen. Und zum Dritten gibt es graphische Werkzeuge wie Denemo (<http://www.denemo.org/>) und NtEd (<http://vsr.informatik.tu-chemnitz.de/staff/jan/nted/nted.xhtml>) (Projekt scheint eingeschlafen), die LilyPond hinter den von ihnen vorgegeben Kulissen einsetzen, ohne daß der Anwender selbst mit LilyPond-Quelltexten in Berührung käme.

Die Benutzerschnittstelle besteht in einer Eingabesprache. Der Anwender nutzt einen Editor<sup>2</sup> seiner Wahl, um Texte in dieser Eingabesprache<sup>3</sup> zu verfassen.

LilyPond wird dann explizit angewiesen, einen Text in dieser Sprache in ein Ausgabeformat nach Wahl des Benutzers zu übersetzen.

Das kann auf der Kommandozeile so schlicht aussehen wie

```
lilypond test.ly
```

nachdem man die Eingabe in der Datei „test.ly“ untergebracht hat.

Dieser Aufruf würde normalerweise PDF- und (falls in der Datei selbst entsprechend angegeben) MIDI-Dateien erzeugen.

Über weitere Kommandozeilenargumente werden neben diesen auch weitere Ausgabeformate zugänglich:

**MIDI** Diese Ausgabe wird unabhängig von den anderen Ausgabeformaten erzeugt. Die dabei erzeugte Ausgabequalität ist zum Korrekturhören und nicht als musikalisch wertvolles Produkt gedacht.

**PostScript** Das ist das Hauptausgabeformat von LilyPond. In früheren Inkarnationen verwendete LilyPond  $\TeX$ , um PostScript-Dateien zu erzeugen. Dies ist nunmehr seit einigen Jahren Geschichte: das Backend „tex“ ist seit 2008 aus LilyPond gestilgt worden und war schon länger optional.

**PDF** wird stets über Aufruf von Ghostscript erzeugt. Da Ghostscript entsprechende Erweiterungen implementiert, können aber zumindest Hyperlinks eingebettet werden.

**SVG** wird direkt geschrieben.

**PNG** als Grafikformat wird von Ghostscript aus PostScript erzeugt.

Bei der Fontunterstützung ist die „Pango“ (<http://www.pango.org/>)-Bibliothek maßgeblich beteiligt.

LilyPond paßt sich sicherlich am besten auf GNU/Linux als Betriebssystem ein, weil sich dort die zahlreichen Abhängigkeiten von weiterer freier Software am unproblematischsten gestalten. Weitere angebotene Versionen laufen aber auch unter MacOSX und Windows. Diese Versionen werden durch den „Grand Unified Builder“

<sup>2</sup>Unter „Editor“ verstehen die Altvordenen eine Art Textverarbeitungssystem, mit dem man rudimentäre Texte aus einzelnen Zeichen zusammstückeln kann. Falls Ihre Postagentur eine Option „HTML abschalten“ beherrscht, können Sie sich eine ungefähre Vorstellung davon machen, wie damit erzeugte Texte aussehen.

<sup>3</sup>Als Beispiel führt die Eingabe { <c' e'>4 d'8( e'8 f'2) } zum Ergebnis



(<http://lilypond.org/gub/>) durch Crosscompilierung auf einem virtuellen Ubuntu-system generiert und enthalten i.a. neben einem systemüblichen Installierer auch die jeweiligen Werkzeuge, auf deren Vorhandensein LilyPond angewiesen ist.

### 3 LilyPonds Arbeitsweise

Von der Eingabedatei bis zum gesetzten Endprodukt durchläuft die Musik mehrere Stadien. An erster Stelle steht hierbei der „Parser“, der die für Menschen lesbare Eingabe in für den Computer handhabbare Scheme-Datenstrukturen übersetzt. Auch wenn LilyPond in maßgeblichen Teilen in der Programmiersprache C++ geschrieben ist, ist die für den Anwender zugängliche Funktionalität im Scheme-Dialekt Guile<sup>4</sup> geschrieben. Kernstück der musikalischen Strukturen ist hierbei die „Music“-Struktur. Wie sähe jetzt das bereits erwähnte Beispiel `{ <c' e'>4 d'8( e'8 f'2) }` aus, wenn man es in Scheme schriebe? Stellt man diesem Beispiel `\displayMusic` voran und läßt LilyPond darauf los, so erhält man auf der Textkonsole die etwas umfängliche Antwort

```
(make-music
  'SequentialMusic
  'elements
  (list (make-music
        'EventChord
        'elements
        (list (make-music
              'NoteEvent
              'duration
              (ly:make-duration 2 0 1 1)
              'pitch
              (ly:make-pitch 0 0 0))
            (make-music
              'NoteEvent
              'duration
              (ly:make-duration 2 0 1 1)
              'pitch
              (ly:make-pitch 0 2 0))))
        (make-music
          'NoteEvent
          'articulations
          (list (make-music
                'SlurEvent
                'span-direction
                -1))
              'duration
              (ly:make-duration 3 0 1 1)
              'pitch
              (ly:make-pitch 0 1 0))
          (make-music
            'NoteEvent
```

<sup>4</sup>Guile (<http://www.gnu.org/software/guile/>), „GNU Ubiquitous Intelligent Language for Extensions“ ist die in den Richtlinien des GNU-Projekts festgeschriebene Erweiterungssprache. Tatsächlich stellt LilyPond mit seiner Nutzung von Guile eher die Ausnahme als die Regel dar; und auch die enge Verzahnung der Erweiterungssprache mit den Interna des Programmes ist beispiellos.

```

'duration
(ly:make-duration 3 0 1 1)
'pitch
(ly:make-pitch 0 2 0))
(make-music
'NoteEvent
'articulations
(list (make-music
      'SlurEvent
      'span-direction
      1))
'duration
(ly:make-duration 1 0 1 1)
'pitch
(ly:make-pitch 0 3 0)))

```

Ins Auge fällt sofort das für Sprachen der Lisp-Familie, zu denen Scheme gehört, typische strukturierende Klammergebirge. Wenn man hineinschaut, sieht man zum einen die für die Erstellung von Music zuständige Funktion `make-music`, zum anderen deren Argumente. Das erste ist der Typ der erstellten Struktur, die weiteren Elemente sind jeweils ein Argumenttyp, gefolgt vom Argument selbst. Das ganze Beispiel ist vom Typ `SequentialMusic`, der Eingangsakkord als erstes Element ist ein `EventChord`, der wiederum aus zwei `NoteEvent` besteht. Tonhöhen werden als `pitch` festgehalten; der erzeugende Aufruf `ly:make-pitch` erhält Oktave, Leiterton und Vorzeichen als Argument. Die Bindebögen verstecken sich als `SlurEvent` in den Noten. Wollte man die Eingabe in dieser Form selbst gestalten, so wäre dies tatsächlich möglich, wenn man diesem Sermon ein `$` voranstellte. Das Resultat würde dann in der Tat von LilyPond in derselben Form interpretiert wie die anwenderfreundlichere Texteingabe.

Nachdem der Parser nun LilyPonds Texteingabe in das für ihn verständlichere Format umgesetzt hat (das tatsächlich eine erheblich kompaktere Form hat als die Klartextdarstellung des Scheme-Codes vermuten ließe), muß dieses interpretiert werden.

LilyPond tut dies in einer Hierarchie von „Contexts“, die sich etwa in „StaffGroup“ (Notensystemgruppen), „Staff“ (Notensysteme, von denen etwa Klaviernoten zwei pro StaffGroup aufweisen, eines für jede Hand), „Voice“ (eine Stimme: Notenhäse beispielsweise werden pro Stimme, die ihrerseits noch Akkorde als Untereinheit enthalten kann, vergeben) gliedert. Tatsächlich gibt es noch eine Vielzahl weiterer Kontexte, etwa zur Darstellung von Liedtexten, Akkordsymbolen, Akkorddiagrammen, Gitarrentabulaturen, Schlagwerknotation und weiterer Elemente.

Die Musik wird innerhalb dieser geschachtelten Kontexte dann von „Iterators“ quasi abgespielt und sendet die enthaltenen Elemente in richtiger zeitlicher Abfolge an „Engraver“ (Graveure), die vorab ihr Interesse an bestimmten musikalischen Elementen („Events“) bekundet haben und diese dann zu Papier bringen. Ihre Vettern sind „Performer“ (Aufführende), die dieselben Events statt in Notenhäse und -köpfe in MIDI-Dateielemente umwandeln.

Nachdem dieser kleine Exkurs uns ganz unversehen in unsägliche Tiefen geführt hat, wird es Zeit, zum Thema des Vortrags zurückzufinden.

#### 4 Präludium #1 in Scheme

Wenn es darum geht, was sich im letzten Jahr an revolutionärem in LilyPond getan hat, ist es lehrreich, ein von Nicolas Sceaux angegebenes Programmierbeispiel (<http://nicolas.sceaux.free.fr/prelude/prelude.html>) heranzuziehen. Seine Experimente auf der Guile-Kommandozeile können wir nachvollziehen, indem wir

```
lilypond scheme-sandbox
```

aufrufen. Tatsächlich wird das dort gegebene Beispiel nicht mehr in der angegebenen Form funktionieren; bringt man aber zum einen das Programm durch den Konvertierungsaufruf

```
convert-ly -ed prelude.ly
```

auf die aktuelle Eingabesyntax und geht die Analyse von Nicolas Sceaux zum anderen schrittweise durch, fällt auf, daß das verbleibende Problem darin besteht, daß die `NoteEvent`-Elemente der einzelnen Noten nicht mehr in `EventChord`-Elemente geschachtelt sind. Damit haben wir dann auch die kontroverseste (<http://lists.gnu.org/archive/html/lilypond-devel/2012-01/msg00716.html>) Änderung, eine Änderung der internen Darstellung, bei der Einzelnoten nicht mehr zwangsweise in einem `EventChord` geschachtelt sind. Diese Inkompatibilität hat eine Reihe praktischer Vorteile, die im wesentlichen dann zum Tragen kommen, wenn man Akkorde synthetisieren möchte. So ist das Beispiel

```
tonika=fis'  
{ <\tonika \transpose c g \tonika> }
```



erst durch den Verzicht auf diese Zwangsverschachtelung praktikabel geworden, weil hier ansonsten ein `EventChord` zu viel aufträte. Um nicht um den heißen Brei zu reden, gehen wir jetzt ungebremst in medias res: auf welche Weise verhelfen Änderungen des letzten Jahres, die von Nicolas Sceaux durchexerzierte komplexe Programmieraufgabe für die vereinfachte Eingabe eines Stückes systematischer Musik handhabbarer zu machen?

```

ph = #(define-music-function (parser location p1 p2 p3 p4 p5)
  (ly:pitch? ly:pitch? ly:pitch? ly:pitch? ly:pitch?)
  #{
    \repeat unfold 2 { $p1 2 } |
    \repeat unfold 2 { r16 $p2 8. ~ $p2 4 } |
    \repeat unfold 2 { r8 $p3 16 $p4 $p5 $p3 $p4 $p5 } |
  #})
\parallelMusic #'(low middle high)
{
  \ph c' e' g' c'' e''
  R1*7 | \skip 1*7 | \oneVoice R1*7 \voiceOne |
  \ph a c' e' g' c''
  \voiceTwo | \change Staff = "down" \voiceOne | \oneVoice |
  \ph d a d' fis' c''
  \oneVoice R1*21 \voiceTwo | \skip 1*21 | R1*21 |
  \ph c, c g bes e'
  c,2~ c, | r16 c8. ~ c4 ~ c2
  | r8 f16 a c' f' c' a c' a f a f d f d |
  c,2~ c, | r16 b,8. ~ b,4 ~ b,2
  | r8 g'16 b' d'' f'' d'' b' d'' b' g' b' d' f' e' d' |
  c,1\fermata | c1 | <e' g' c''>1\fermata \bar "|." |
}
\score {
  \new PianoStaff <<
    \new Staff = "up" {
      \compressFullBarRests \accidentalStyle modern
      << \high \middle >>
    }
    \new Staff = "down" { \clef bass \low }
  >>
}

```

The image shows a musical score for piano, consisting of two systems of music. Each system has two staves: an upper staff in treble clef and a lower staff in bass clef. The first system starts with a treble clef staff containing a series of eighth notes and rests, followed by a bass clef staff with a single note. The second system begins with a treble clef staff containing a series of eighth notes and rests, followed by a bass clef staff with a single note. The score includes various musical notations such as notes, rests, and fermatas.



Analysieren wir das hier vorliegende Exzerpt, so fällt zunächst auf, daß ein Hauptthema in der Vorlage hier vollkommen entfällt: Nicolas verwendet beträchtlichen Aufwand darauf, aus dem Eingabematerial Tonhöhen zu extrahieren. Die hier verwendete Musikfunktion `\ph` allerdings deklariert ihre 5 Eingangsvariablen von vorneherein als `ly: pitch?`, also als Tonhöhe und kann sich deswegen ersparen, in den Tiefen von Musik nach dieser Information zu graben. Diese Möglichkeit existiert erst seit etwa einem halben Jahr.

Statt mit aufwendiger Analyse ist `\ph` gleich mit Synthese beschäftigt und verwendet dazu das auch früher schon einsetzbare Konstrukt `#{. . .#}`. Die drei generierten Zeilen entsprechen Unterstimme (ein Liegeton von einer halben Note mit der ersten Tonhöhe), Mittelstimme (nach einer Sechzehntelpause eine punktierte Achtel von der zweiten Tonhöhe, angebunden an eine Viertel derselben Tonhöhe), Oberstimme (dem Leser zur Analyse überlassen).

Das hier eingesetzte Konstrukt `\parallelMusic` verteilt den Quelltext an den Taktstrichen `|` auf die drei unterschiedlichen Stimmen `low`, `middle` und `high`. Die weiteren Elemente im Quelltext sind nicht von dem verschieden, was auch schon vor Jahren möglich gewesen wäre: `\oneVoice` ist für Stimmen, die ihr Notensystem für sich alleine haben, `\voiceOne` und `\voiceTwo` richten Notenhälse und andere Konstrukte in einer Stimme jeweils so ein, daß zwei Stimmen in ein System passen. Die Mittelstimme wechselt mit `\change Staff` innerhalb des Stückes vom oberen in das untere Notensystem.

Diese und die weiteren Elemente sind Konstrukte, die im auch online verfügbaren Handbuch (<http://lilypond.org/doc/v2.14/Documentation/notation/>) älterer Versionen ersichtlich sind und sollen hier nicht weiter erläutert werden.

## 5 Und Fuge...

Um diesen Beitrag zu vervollständigen, folgt noch ein Durchgang durch das Thema der Erweiterungen, der weitere Entwicklungen nennt. Der Leser mag nicht enttäuscht sein, wenn er nicht gleich alle Details erfassen kann: diese Art der Musik erfordert ein geübtes Ohr, daß sich nach längerer Beschäftigung mit der Materie einstellt:

- Musikfunktionen können auch optionale Argumente enthalten. Als Resultat sind folgende Funktionen aus dem im Parser festverdrahteten Wortschatz von

LilyPond verschwunden und zu Musikfunktionen geworden. Das bedeutet, daß der Benutzer selbst Funktionen mit vergleichbarer Eingabesyntax definieren könnte:

```
\grobdescriptions, \key, \mark, \once, \partial, \relative, \skip,  
\time, \times, \transpose
```

Der Quelltext von `ly/music-functions-init.ly` (<http://git.savannah.gnu.org/cgi/lilypond.git/tree/ly/music-functions-init.ly>) enthält die entsprechenden Definitionen.

- Das Konstrukt `#{ . . #}` kann nicht nur eine Musiksequenz enthalten, sondern auch (Zitat in Englisch):

single music events, void music expressions, post events, markups (mostly freeing users from having to use the markup macro), markup lists, number expressions, context definitions and modifications, and a few other things.

Innerhalb dieses Konstrukts enthaltene Scheme-Ausdrücke, die mit `$` oder `#` eingeleitet werden, sind in der „lexical closure“ des umgebenden Scheme-Ausdrucks, können also lokale Variablen der umgebenden Funktion ohne spezielle Maßnahmen referenzieren.

- mit `\define-event-function` und `\define-scheme-function` können, in Analogie zu den schon lange existierenden „music functions“ in Scheme Funktionen für „Postevents“ und beliebige Ausdrücke eingesetzt werden, die sich nahtlos in die LilyPond-Syntax einfügen.

Tatsächlich ist die Liste der Änderungen, die in der letzten Zeit stattgefunden haben, erheblich länger: zahlreiche Verbesserungen in der Qualität der Ausgabe haben ebenfalls Eingang gehalten, und eine Vielzahl weiterer Funktionalität ist hinzugekommen.

Dieser Beitrag konzentrierte sich nur auf wesentliche Änderungen, die das Programmiermodell von LilyPond betreffen und damit einen Eindruck davon geben, welche Funktionalität in diesem Bereich für „Normalsterbliche“ in Reichweite gerückt ist.

Das Projekt LilyPond kann neue Mitarbeiter gut gebrauchen: im Bereich der Dokumentation, der Übersetzung, und nicht zuletzt auch der Erweiterung. Und gerade letzteres sollte in der letzten Zeit um einiges leichter geworden sein.

# MapReduce – Parallelität im Großen und im Kleinen

Jens Lang

jens.lang@informatik.tu-chemnitz.de

MapReduce ist ein Programmiermodell für Parallelrechner, das die Möglichkeit bietet, effiziente parallele Programme mit geringem Aufwand zu implementieren. „MapReduce“ bedeutet, dass man ein Problem in einen Map- und einen Reduce-Teil aufteilt und jeweils eine Map- und eine Reduce-Funktion implementiert. Dadurch lässt sich eine erstaunliche Vielzahl von Problemen effizient mit nur diesen beiden Schritten lösen. Außerdem wird das Problem implizit in viele unabhängige Teilprobleme zerlegt, sodass es gut parallelisiert werden kann.

Neben der geringen Komplexität des Programmiermodells, die das Programmieren erleichtert, bietet MapReduce den Vorteil, dass die Bibliothek automatisch für gleichmäßige Verteilung der Last auf die Rechenknoten, für das Umsortieren der Daten, für Kommunikation, Fehlertoleranz etc. sorgt. Der Programmierer muss sich um nichts als seinen Algorithmus kümmern.

## 1 Einleitung

*MapReduce* ist ein Konzept für die parallele Programmierung, das Programme in zwei Phasen einteilt: eine Map-Phase und eine Reduce-Phase. Für jede dieser Phasen definiert der Benutzer jeweils eine Funktion. Die dadurch im Vergleich zu anderen Ansätzen stark reduzierte Komplexität ermöglicht es auch Programmierern mit wenig Erfahrung im parallelen oder verteilten Programmieren, parallele Programme zu entwickeln. Um die Verteilung der Daten, ihre Umverteilung zwischen den Phasen, Kommunikation zwischen Rechenknoten, Lastbalancierung und ggf. Fehlertoleranz muss sich der Programmierer nicht sorgen.

Die Funktionen *map* und *reduce*, die die beiden Phasen bestimmen, entstammen in ihrer Semantik der funktionalen Programmierung. Ansätze, ihr Prinzip zur Parallelisierung von Programmen zu nutzen gab es schon in den neunziger Jahren, z. B. [5].

Erstmals derart formuliert, dass es sich für einen breites Spektrum von Algorithmen praktisch nutzen lässt und gleichzeitig automatisch eine parallele Abarbeitung der Programme ermöglicht, wurde das Prinzip MapReduce in einem Artikel von Google-Mitarbeitern im Jahr 2004 [6, 7]. Das Programmiermodell wurde im Zuge der Entwicklung einer entsprechenden Bibliothek für die interne Verwendung entwickelt und schließlich veröffentlicht. Zielsetzung war es, eine Bibliothek zu schaffen, die den Entwurf und die Ausführung von Programmen für Cluster bestehend aus tausenden von Rechenknoten vereinfacht.

Programme können im MapReduce-Modell vollständig unabhängig von der Hardwarekonfiguration, die bei ihrer Ausführung zur Verfügung stehen wird, formuliert werden. Daher ist ein Algorithmus, der in der MapReduce-Notation ausdrückbar ist,

leicht auf verschiedene Hardwareplattformen wie Multicore-Rechner, Cluster, Grafikprozessoren usw. portierbar. Da die Aufrufe der Map-Funktion bzw. der Reduce-Funktion jeweils unabhängig voneinander sind, sind MapReduce-Programme hochgradig parallelisierbar und skalieren sehr gut: Sie können sowohl auf Multicore-Rechnern mit nur wenigen Prozessoren als auch in Clustern oder in Clouds mit hunderten oder tausenden von Knoten effizient ausgeführt werden.

## 2 Das MapReduce-Programmiermodell

MapReduce arbeitet auf einem großen Vektor (Array) von Eingabedaten. Diese haben die Struktur *Schlüssel*→*Wert*, wobei die Schlüssel keineswegs eindeutig sein müssen. Die vom Nutzer definierte Funktion *map* wird auf jedes Element des Arrays angewandt. Jeder Aufruf produziert jeweils wieder ein oder mehrere Schlüssel-Wert-Paare, die sogenannten *Intermediate-Paare*. Es können sich sowohl die Schlüssel als auch die Werte ändern. Wichtig ist, dass die Aufrufe der Map-Funktion vollkommen unabhängig voneinander arbeiten, es wird also kein interner Zustand gespeichert.

Als nächstes werden alle Intermediate-Paare nach ihren Schlüsseln gruppiert. Werte mit gleichem Schlüssel werden in einem Vektor zusammengefasst, der diesem zugeordnet wird. Mit diesen Schlüssel-Wertevektor-Paaren wird die Reduce-Funktion aufgerufen, die ebenfalls vom Benutzer definiert wird. Reduce ist aus funktionalen Programmiersprachen zum Teil auch unter der Bezeichnung *fold* bekannt und bewirkt in seiner ursprünglichen Bedeutung eine *Reduktion* der *n* Elemente eines Vektors in ein Skalar. Diese Semantik wurde von einigen Bibliotheken für die parallele Programmierung wie MPI oder OpenMP aufgegriffen, wo man mittels *reduce* aus einem Vektor z. B. die Summe seiner Elemente oder ihr Maximum berechnen kann. In MapReduce ist es auch zulässig, dass die Reduce-Funktion überhaupt kein Ergebnis zurückliefert.

Der Originalartikel [6] gibt keine exakte Definition der Syntax der Map- und Reduce-Funktion. [10] liefert einen Vorschlag für die Deklaration von MapReduce in Haskell:

```
mapReduce :: forall k1 k2 v1 v2 v3
=> (k1 -> v1 -> [(k2,v2)]) -- Map-Funktion
-> (k2 -> [v2] -> Maybe v3) -- Reduce-Funktion
-> Map k1 v1                -- Eingabe
-> Map k2 v3                 -- Ausgabe
```

Es wird jeweils eine benutzerdefinierte Map- und Reduce-Funktion übergeben. Die Map-Funktion hat einen Schlüssel *k1* und einen Wert *v1* als Eingabe und einen Vektor mit Schlüssel-Wert-Paaren [(*k2*, *v2*)] (ggf. eines anderen Typs) als Ausgabe. Die bei den Aufrufen der Map-Funktion entstandenen Vektoren [(*k2*, *v2*)] bilden die Intermediate-Paare. Die Reduce-Funktion hat einen Schlüssel *k2* und den Vektor von gruppierten Werten [*v2*], die alle den gleichen Schlüssel besitzen, als Eingabe. Der Wert *v3*, bei dem es sich ebenfalls um einen Vektor handeln kann, bildet die Ausgabe der Reduce-Funktion. Das Schlüsselwort *Maybe* deutet an, dass es auch möglich

ist, kein Ergebnis zurückzuliefern. Die Eingabe für MapReduce ist eine Map-Datenstruktur, also ein assoziatives Array, die jedem Schlüssel  $k_1$  einen Wert  $v_1$  zuweist. Die Ausgabe von MapReduce ist ebenfalls eine Map-Datenstruktur, jedoch mit einem neuen Typ von Schlüssel  $k_2$  und Wert  $v_3$ .

Durch die Aufteilung in die Map- und die Reduce-Phase existiert eine implizite Barriere zwischen beiden. Hier werden die Intermediate-Paare von ihren Map-Knoten auf die entsprechenden Reduce-Knoten (die physikalisch auf den gleichen Rechenknoten liegen können) umsortiert. Einige MapReduce-Implementierungen sortieren die Schlüssel gleichzeitig nach einer anzugebenden Ordnungsrelation, andere hingegen verzichten aus Performancegründen darauf.

Eine zusätzliche Indirektionsstufe, wie sie auch durch die MapReduce-Bibliothek eingeführt wird, verringert zwar unter Umständen die Komplexität der Programme, erhöht aber in vielen Fällen auch deren Laufzeit. Für die Phoenix-Bibliothek [12], die später noch genauer vorgestellt wird, wurde die Laufzeit verschiedener Anwendungen untersucht, die einmal mit MapReduce und einmal mit PThread parallel implementiert wurden. Für viele Anwendungen gab es keinen wesentlichen Unterschied in der Laufzeit, bei einigen war jedoch die MapReduce-Implementierung deutlich langsamer. Inzwischen existiert eine Folgeversion von Phoenix [13], deren Geschwindigkeit im Vergleich zur Ausgangsversion deutlich verbessert sein soll, sodass sogar dieser Unterschied möglicherweise entfällt.

### 3 Beispiele

Die Verwendung des MapReduce-Programmiermodells soll anhand zweier kleiner Beispiele verdeutlicht werden: Das erste Beispiel berechnet Kräfte zwischen im Raum liegenden geladenen Teilchen. Das zweite erzeugt einen Index für eine Menge von Textdateien.

#### 3.1 Kraftberechnung

Das Beispiel zur Kraftberechnung wird in Algorithmus 1 im Pseudocode dargestellt. Für ein System von (fest im Raum liegenden) geladenen Teilchen  $p_q$  berechnet es die Kräfte, die auf Probeteilchen  $p_s$  wirken, die ebenfalls geladen sind und in einem festen Punkt im Raum liegen. Jedes Teilchen  $p$  soll die Attribute  $p.pos$  für seine Position im Koordinatensystem,  $p.charge$  für seine Ladung und  $p.id$  als eindeutige Kennzahl besitzen.

Die Map-Funktion arbeitet wie folgt: Der Eingabeschlüssel wird ignoriert, der Eingabewert ist ein Kraft „aussendendes“ Quellteilchen  $p_q$ . Nun wird für alle Senkenteilchen  $p_s$ , die Kraft „empfangen“, die Kraft berechnet, mit der  $p_q$  auf sie wirkt. Dafür wird zunächst mit der Funktion `distance()` ihr Abstand bestimmt. Aus dem Abstand

---

**Algorithmus 1** MapReduce-Beispiel zur Kraftberechnung

---

```
procedure map(key =  $\emptyset$ , value =  $p_q$ )
  for all  $p_s$  do
     $d := \text{distance}(p_q.\text{pos}, p_s.\text{pos})$ 
     $\vec{F} := \text{coulomb}(p_q.\text{charge}, p_s.\text{charge}, d)$ 
    emit_intermediate(key =  $p_s.\text{id}$ , value =  $\vec{F}$ )
  end for
end procedure
procedure reduce(key = id, values =  $\vec{F}[]$ )
   $\vec{F}_{tot} := 0$ 
  for all  $\vec{F}_i \in \vec{F}[]$  do
     $\vec{F}_{tot} := \vec{F}_{tot} + \vec{F}_i$ 
  end for
  emit(value =  $\vec{F}_{tot}$ )
end procedure
```

---

und den beiden Ladungen wird mit der Funktion `coulomb()` nach der bekannten Formel die wirkende Coulomb-Kraft berechnet. Als Intermediate-Paare werden jeweils die Kennzahl des Senkenteilchens  $p_s$ , auf das die Kraft wirkt, und die angreifende Kraft ausgegeben.

In der Zwischenphase werden alle Kräfte, die am gleichen Senkenteilchen wirken, zu einem Vektor zusammengefasst, die diesem zugeordnet werden. Diese Schlüssel-Wertevektor-Paare werden an die Reduce-Funktion übergeben.

Anhand der Kennzahl, die den Schlüssel bildet, erkennt die Reduce-Funktion, welches Senkenteilchen  $p_s$  gerade bearbeitet wird. Im Wertevektor stehen alle Kräfte, die an diesem Teilchen angreifen. Alle diese Kräfte werden nun zur angreifenden Gesamtkraft  $\vec{F}_{tot}$  aufsummiert. Zum Schluss wird diese Gesamtkraft als Ergebnis ausgegeben. Die Kennzahl des betrachteten Senkenteilchens wird ihr hierbei implizit zugeordnet.

### 3.2 Indexerzeugung

Das zweite Beispiel soll einen Satz Textdateien einlesen und in einem Index speichern, welches Wort in welchen Dateien vorkommt. Es wird im Pseudocode in Algorithmus 2 gezeigt.

Die Map-Funktion erhält jeweils eine Textdatei als Eingabewert und den dazugehörigen Dateinamen als Schlüssel. Der Dateinhalt wird an Leerzeichen und Interpunktionen in Worte geteilt. Schließlich wird für jedes Wort ein Intermediate-Paar mit dem Wort als Schlüssel und dem Dateinamen als Wert generiert.

---

## Algorithmus 2 MapReduce-Beispiel für die Erstellung eines Index

---

```
procedure map(key = filename, value = text (file content))
  for each word in text do
    emit_intermediate(key = word, value = filename)
  end for
end procedure
procedure reduce(key = word, values = filenames[])
  remove double entries in filenames[]
  emit(value = filenames[])
end procedure
```

---

Die Reduce-Funktion hat als Eingabeschlüssel ein Wort und als Eingabewertevektor eine Liste der Namen der Dateien, in denen dieses Wort vorkommt. Sie entfernt doppelte Dateinamen und gibt die bereinigte Liste der Dateinamen als Ergebnis aus. Man erhält also als Ergebnis eine Zuordnung Wort→Dateinamen.

## 4 Optimierungen

Die meisten Bibliotheken verwenden verschiedene Optimierungen, um die Ausführung von MapReduce-Programmen zu beschleunigen. Viele von diesen ähneln sich im Prinzip. Eine Auswahl soll in diesem Abschnitt vorgestellt werden.

Werden in der Map-Phase sehr viele Intermediate-Paare erzeugt, müssen für die Reduce-Phase entsprechend viele Daten umsortiert werden. Besonders bei physikalisch voneinander getrennt liegenden Speicherorten ist das aufwändig. Eine **Combiner-Funktion** bietet eine Möglichkeit, die Daten bereits auf den Map-Knoten etwas zu reduzieren. Häufig arbeitet sie analog zur „realen“ Reduce-Funktion.

Die **Splitter-Funktion** dient dazu, die Daten anhand ihrer Eingabeschlüssel möglichst gleichmäßig auf die Map-Knoten zu verteilen. Sinnvoll kann es auch sein, Daten mit vermutlich gleichem Reduce-Schlüssel schon in der Map-Phase auf dem gleichen Knoten zu verarbeiten, um die Menge der umzuverteilenden Daten möglichst gering zu halten. Wird keine Splitter-Funktion angegeben, findet die Bibliothek eine andere Methode zur gleichmäßigen Verteilung der Daten

Die **Partition-Funktion** bestimmt die Zuordnung von Schlüsseln zu Knoten für die Reduce-Phase. Auch hier sollte auf eine möglichst gleichmäßige Lastverteilung geachtet werden. Ist keine Partition-Funktion vorhanden, kann die Verteilung z. B. anhand einer Hashfunktion festgelegt werden.

Bei mehr als 1000 Knoten ist der Ausfall eines Knotens eher die Regel als die Ausnahme. Fehlertoleranz ist daher ein wichtiges Merkmal von Anwendungen für solche Systeme. Neben dem Totalausfall der Hardware soll auch die verzögerte Ausführung von Tasks abgedeckt werden. In MapReduce werden hierfür üblicherweise **Backup-**

**Tasks** verwendet. Sie werden gestartet, wenn nach einer bestimmten Zeit noch keine Antwort von dem verantwortlichen Knoten eingetroffen ist. Die Rechenlast wird zwar nun für diesen Task verdoppelt, es ergibt sich aber eine deutlich verkürzte Ausführungszeit, wenn nicht auf den letzten „Nachzügler“ noch gewartet werden muss. Das Ergebnis des langsameren Knotens wird einfach verworfen.

Unter Umständen ist es erforderlich, einen Strom kontinuierlich anfallender Daten zu verarbeiten. Dafür gibt MapReduce-Bibliotheken, die eine so genannte **barrierefreie Ausführung** bieten, z. B. [4]. Hier wird die Reduce-Funktion immer in bestimmten Abständen aufgerufen und verarbeitet die bis dahin angefallenen Daten. Dieses Verfahren kann beispielsweise auch dafür eingesetzt werden, dass man schon nach der Verarbeitung von 10 % der Daten erste Ergebnisprognosen aufstellen kann.

## 5 MapReduce-Bibliotheken für verschiedene Plattformen

MapReduce ist für alle wichtigen Plattformen verfügbar. Die populärsten MapReduce-Bibliotheken sollen in diesem Abschnitt vorgestellt werden.

**Googles MapReduce-Bibliothek** [6,7] ist die erste Implementierung des MapReduce-Programmiermodells. Sie wird im Google-Datencenter unter anderem für den Aufbau des Google-Suchmaschinenindex, für die statistische maschinelle Übersetzung von Google Translate, für den Nachrichtenaggregator Google News und viele weitere Dienste von Google verwendet. Nach [6] hat sich gezeigt, dass mit der Verwendung des MapReduce-Programmiermodells bei vielen Programmen die Komplexität erheblich verringert und die Wartbarkeit verbessert werden konnte.

Google verarbeitet mit MapReduce-Anwendungen Datenmengen in Petabyte-Größenordnung. Diese werden in dem verteilten Dateisystem GoogleFS [8] gespeichert, das in der Regel jeweils drei Kopien jedes Datensatzes vorhält. Die Datenverteilung wird bei der Zuweisung von Tasks an Rechenknoten berücksichtigt.

**Hadoop** [1] ist die „Open-Source-Antwort“ auf Googles MapReduce. Eine der treibenden Kräfte zu Beginn war der Suchmaschinenanbieter Yahoo. Heute liest sich die Liste der Hadoop-Nutzer wie das Who-is-Who des Silicon Valley [2]: Facebook, Amazon, Ebay, Twitter und viele mehr setzen Hadoop ein. Der Quelltext ist unter dem Dach der Apache Software Foundation angesiedelt.

Hadoop basiert auf Java, es existieren aber z. B. auch Bindings für Python. Als Dateisystem kommt HDFS zum Einsatz, das sich ebenfalls an das GoogleFS anlehnt. Hadoop ist für den Einsatz in verteilten Systemen konzipiert. Installiert wird es in der Regel in großen Clustern, die auch aus mehreren tausend Knoten bestehen können. Nutzer senden ihre Hadoop-Jobs an ein Verwaltungssystem, welches diese in den Abarbeitungsplan einstellt und zu gegebener Zeit ausführt. Da nicht jeder Job alle Knoten benutzt, können mehrere Jobs parallel verarbeitet werden. Aufwändige

Jobs können hingegen auch eine entsprechend große Anzahl an Rechenknoten erhalten. Für den kommerziellen Einsatz existieren Cloud-Anbieter, die eine Hadoop-Umgebung bereitstellen, bei der – wie im Cloud-Computing üblich – nach verbrauchter Rechenleistung abgerechnet wird.

**Phoenix** [13] ist eine MapReduce-Bibliothek für Parallelrechner mit gemeinsamem Speicher, also Multicore-Cluster, SMP-Systeme und ähnliches. Phoenix ist in C geschrieben und arbeitet auf PThread-Basis. Es bietet die Möglichkeit, Combiner-, Splitter- und Partition-Funktionen anzugeben. Das Sortieren der Schlüssel nach der Reduce-Phase erfolgt hier optional.

**MR-MPI** [11] ist eine MapReduce-Bibliothek, die MPI zur Kommunikation zwischen den Knoten verwendet. Sie ist also gut geeignet für den Einsatz auf Clustern mit verteiltem Speicher, z. B. im Wissenschaftlichen Rechnen. Hier wird MapReduce nicht nach der reinen Lehre implementiert, sondern der Nutzer kann die Map- und die Reduce-Funktion jeweils einzeln aufrufen oder z. B. auch zuerst zwei Map-Funktionen und dann eine Reduce-Funktion. Funktionalität zum Sortieren nach Schlüsseln oder Werten und zur feingranularen Beeinflussung der Verteilung der Daten auf die Knoten wird bereitgestellt.

**Mars** [9] ist die bekannteste MapReduce-Bibliothek für Grafikprozessoren. Programme für Mars werden in CUDA geschrieben. Die besondere Schwierigkeit besteht hier, dass Grafikprozessoren keine dynamische Speicherallokation beherrschen und nicht direkt auf die Festplatte zugreifen können.

Das **Qt-Framework** [3] stellt mit der *QtConcurrent*-Bibliothek eine MapReduce-artige Funktionalität bereit. Hier können jedoch keine Schlüssel angegeben werden, sondern nur Werte. Es wird also nur eine einzige Instanz der Reduce-Funktion aufgerufen. Während die Map-Funktion parallel abgearbeitet wird, arbeitet die Reduce-Funktion sequentiell.

## 6 Zusammenfassung

MapReduce wurde in seiner Erstveröffentlichung als Methode vorgestellt, mit der man ohne große Komplexität parallele Programme ausdrücken und effizient ausführen kann. Dass dies gelungen ist, zeigen die zahlreichen Bibliotheken für die verschiedenen Plattformen, die das Prinzip aufgegriffen und teilweise adaptiert haben.

MapReduce reduziert die Komplexität des parallelen Programmierens, sodass sich Programme übersichtlicher, schneller und besser wartbar implementieren lassen. Mittels verschiedenen Optimierungen laufen viele MapReduce-Programme ähnlich effizient wie mit anderen Methoden implementierte Programme.

## Literatur

- [1] *Apache Hadoop*. <http://hadoop.apache.org/>.
- [2] *Hadoop Wiki – Powered By*. <http://wiki.apache.org/hadoop/PoweredBy>.
- [3] *Qt – cross-platform application and UI framework*. <http://qt.nokia.com>.
- [4] Condie, Tyson, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, John Gerth, Justin Talbot, Khaled Elmeleegy und Russell Sears: *Online aggregation and continuous query support in MapReduce*. In: *Proceedings of the 2010 international conference on Management of data, SIGMOD '10*, Seiten 1115–1118. ACM, 2010.
- [5] Darlington, John, Yi-ke Guo, Hing Wing To und Jin Yang: *Parallel skeletons for structured composition*. *SIGPLAN Notices*, 30:19–28, August 1995.
- [6] Dean, Jeffrey und Sanjay Ghemawat: *MapReduce: simplified data processing on large clusters*. In: *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, Seiten 10–10. USENIX Association, 2004.
- [7] Dean, Jeffrey und Sanjay Ghemawat: *MapReduce: simplified data processing on large clusters*. *Communications of the ACM*, 51:107–113, Januar 2008.
- [8] Ghemawat, Sanjay, Howard Gobioff und Shun-Tak Leung: *The Google file system*. *SIGOPS Operating Systems Review*, 37:29–43, Oktober 2003.
- [9] He, Bingsheng, Wenbin Fang, Qiong Luo, Naga K. Govindaraju und Tuyong Wang: *Mars: a MapReduce framework on graphics processors*. In: *Proceedings of the 17th international conference on Parallel architectures and compilation techniques, PACT '08*, Seiten 260–269. ACM, 2008.
- [10] Lämmel, Ralf: *Google's MapReduce programming model — Revisited*. *Science of Computer Programming*, 68:208–237, Oktober 2007.
- [11] Plimpton, Steven J. und Karen D. Devine: *MapReduce in MPI for Large-scale graph algorithms*. *Parallel Computing*, 37(9):610–632, 2011.
- [12] Ranger, Colby, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski und Christos Kozyrakis: *Evaluating MapReduce for Multi-core and Multiprocessor Systems*. In: *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, Seiten 13–24. IEEE, 2007.
- [13] Talbot, Justin, Richard M. Yoo und Christos Kozyrakis: *Phoenix++: modular MapReduce for shared-memory systems*. In: *Proceedings of the second international workshop on MapReduce and its applications, MapReduce '11*, Seiten 9–16. ACM, 2011.

# Performancemessungen und -optimierungen an GNU/Linux-Systemen

Wolfram Luithardt, Daniel Gachet und Olivier Nasrallah

Hochschule für Technik und Architektur Freiburg, Boulevard de Pérolles 80,

CH-1705 Fribourg, Schweiz

## 1 Einleitung

Die Performance eines System ist natürlich ein extrem wichtiger Faktor für die Entscheidung, ob dieses System für eine bestimmte Aufgabe eingesetzt werden kann oder nicht. Dies gilt sicherlich für eingebettete Systeme noch mehr als für Desktops bzw. Laptops, da bei ersteren die Aufgabe meistens ja bereits bei der Konzeption des Systems recht gut spezifiziert ist und dadurch eine wesentlich bessere Entscheidungsgrundlage vorliegt. Es ist also wichtig, berechnen oder zumindest abschätzen zu können, ob ein geplantes System später den Aufgaben gewachsen ist, oder ob es zu Performance-engpässen kommen kann. Dabei ist zu beachten, dass heutige Computersysteme die eigentliche Performance häufig nur kurzzeitig beim Auftreten bestimmter externer Ereignisse zur Verfügung stellen müssen; ansonsten machen sie relativ wenig. Dies kann z.B. graphisch durch die Systemüberwachungstools beobachtet werden, die ja heute Bestandteil der meisten Distributionen sind (z.B. die Systemüberwachung von gnome).

Für die schnelle Abarbeitung von Ereignissen sollte das System also kurzzeitig die gesamte Performance den entsprechenden Prozessen zur Verfügung stellen können. In diesem Beitrag werden wir einige Methoden vorstellen, wie ein Programmierer ein Programm so gestalten kann, dass einzelnen Prozessen im entscheidenden Moment soviel Ressourcen erhalten, dass die entsprechenden Events möglichst schnell abgearbeitet werden. Damit können Flaschenhälse im Programmablauf verhindert werden, die sonst nur durch eine stärkere Hardware mit z.B. höherer Taktfrequenz (und damit auch höherem Preis, höherem Stromverbrauch usw.) vermeidbar wären. Hierzu ist allerdings ein vertieftes Verständnis der Funktionalitäten vom GNU/Linuxsystem notwendig.

In Kapitel 2 werden wir die Möglichkeiten zur hochgenauen Messung von Zeitdauern diskutieren, bevor wir dann verschiedene Möglichkeiten zur

Optimierung besprechen: In Kapitel 3 das Arbeiten mit Prioritäten, in Kapitel 5 durch das Ändern der Schedulingpolicy, das Optimieren von Speicherzugriffen in Kapitel 6 und das Binden von Prozessen an Prozessoren in Multicore-Systemen in Kapitel 7. In Kapitel 4 werden wir ein Modell entwickeln, mit dem ein System massiv mit vielen pseudo-parallel ablaufenden Prozessen belastet wird.

Die Untersuchungen und Messungen wurden auf einem Ubuntu System (10.04 LTS - Lucid Lynx) mit Linux-Kernel Version 2.6.32-37 durchgeführt. Der Prozessor war ein Intel Core 2 Duo CPU T6400 mit 2.00GHz Taktfrequenz. Der PC war mit 3GB RAM bestückt.

## 2 Hochgenaue Messung von Zeiten

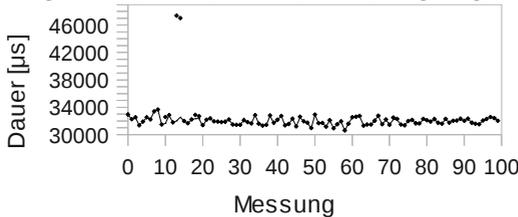
In ursprünglichen UNIX-Systemen konnten die Dauer von Routinen über das times-Interface gemessen wurden. Dafür standen sowohl der UNIX-Kommandozeilenbefehl `times` zur Verfügung, der die Zeiten des darauf folgenden Programms ausgibt, als auch die Systemfunktion `times(..)`, welche eine Struktur vom Typ `tms` zurück gibt, welche verschiedene Zeitressourcen beinhaltet (user time und system time des Prozesses sowie dessen Kinderprozessen). Leider besitzt diese Struktur nur eine Auflösung von 10ms, d.h. die Zeitgrößen werden nur in Portionen von 10ms angegeben. In den 70-er und 80-er Jahren bei einer Taktfrequenz der Prozessoren von einigen wenigen MHz war dies sicherlich ausreichend, heute allerdings ist dies nicht mehr zufriedenstellend. Die 10 ms Auflösung ist durch die Tatsache begründet, dass in Unix-Systemen die Zeitdauer, die einem Prozess zur Verfügung gestellt wird, bevor geprüft wird, ob ein anderer Prozess nun die Ressourcen bekommt, meist bei 10 ms liegt. Dieser Wert ist bei eingebetteten Systemen heute noch weitgehend verwendet, auf Desktop-Systemen liegt er hingegen häufig bei 1ms. An der Auflösung des times-Interfaces ändert dies jedoch nichts.

POSIX definierte dann ein neues, besseres Interface, das eine Zeitauflösung bis max. 1 ns erlaubt. Auf vielen Systemen wird in der Tat auch wirklich Nanosekunden-Auflösungen zurückgegeben, was jedoch nicht zwingend heißen muss, dass die Genauigkeit ebenfalls bei 1 ns liegt. Je nachdem, wann die die Funktion aufgerufen wird, kann die gemessene Zeit einige 10-100ns variieren. Im folgenden

```
clock_gettime(CLOCK_REALTIME, &rt1);
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts1);
for(i=0; i<10000000; i++){
    counter++;
}
clock_gettime(CLOCK_REALTIME, &rt2);
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts2);
```

kleinen Teilprogramm wird dieses POSIX-Interface verwendet um die Länge einer einfachen Schleife zu messen. Die Variable `counter` ist dabei static deklariert um zu vermeiden, dass der Compiler Optimierungen vornimmt, die die Messungen verfälschen können.

rt1 und rt2 bzw. ts1 und ts2 sind Strukturen vom Typ timespec, die Nanosekunden-Auflösungen abspeichern können. Die Verwendung der Posix-times benötigt das Hinzulinken der 'realtime library rt' mit der Option -lrt beim gcc. In der folgenden Abbildung sind 100 Messungen mit diesem Loop gezeigt. Dabei sind die Messungen von (rt2-rt1) sowie (ts2-ts1) aufgetragen. rt (Punkte) gibt die Gesamtzeit an, ts die CPU-Zeit, die für genau diesen Prozess verwendet wird (durchgezogene Linie). Wie leicht zu sehen ist, folgt die Real-time der Loop-Time abgesehen von nur 2 Messungen sehr gut. Dies bedeutet, dass der Prozess die gesamte Ressource



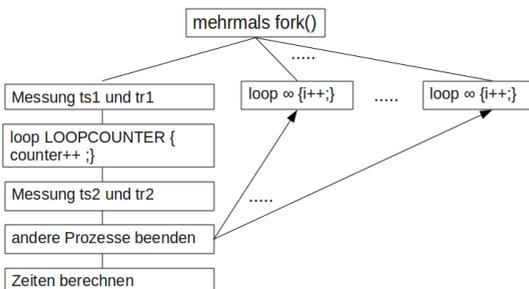
„Processor“ für die meisten Messungen vollständig zur Verfügung hat. Nur bei Messung 13 und 14 scheint ein Hintergrundprozess den Loop für einige Millisekunden unterbrochen zu haben, was dazu führte, dass die Real-Time länger ist als die Prozesszeit (Zeit des Loops). Noch etwas kann aus dieser einfachen Messung abgelesen werden: Die Schwankungen zwischen den einzelnen Messungen liegen in der Größenordnung von  $\pm 1\text{ms}$  (Standardabweichung = 528 Mikrosekunden). Die Gründe hierfür liegen natürlich nicht nur in der Ungenauigkeit der Zeitmessung, sondern vielmehr darin, dass ein komplexes Prozessor- und Betriebssystem wie die x86 Familie und Linux allerhand Mechanismen im Hintergrund besitzt um zwar die Performance zu optimieren aber damit auch den zeitlichen Determinismus zu reduzieren. Genannt seien hier nur Interruptroutinen, die durch zeitliche oder äussere Ereignisse angestoßen werden und Daemons (z.B. kswapd) die Überprüfen, ob genug physikalischer Speicher zur Verfügung steht, usw. Aber auch hardwareoptimierte Speicherzugriffsmechanismen wie den Translation Lookaside Buffer (TLB) und Cachespeicher liefern ihren Beitrag dazu, dass der Ablauf eines Programmes meist nichtdeterministisch ist. Linux bietet über das sys- und das proc-Filesystem sehr interessante Interfaces, solche Methoden zu analysieren und zu beeinflussen. Leider würde es den Rahmen dieses Artikels sprengen, näher darauf einzugehen. Um solche Effekte zu minimisieren wird in der Praxis versucht, die Looplänge so gross zumachen, dass die Effekte relativ zur Gesamtprogrammlänge vernachlässigbar sind. Je kürzer die Programmlaufzeit desto grösser der Einfluss dieser Effekte und desto schwieriger (aber nicht unbedingt uninteressanter :-)) die Interpretation der Daten.

Eine noch bessere (und vor Allem noch genauere) Methode zur Messung von kurzen Ablaufzeiten ist natürlich das Setzen bzw. Zurücksetzen von digitalen Ausgängen und das Messen mit einem Logikanalysator. Hier liegen die Genauigkeiten heutiger Systeme bei wenigen Nanosekunden, während sie beim hier beschriebenen POSIX-Times Interface eher in der Größenordnung von mehreren Mikrosekunden liegen.

### 3 Ändern der Prioritäten

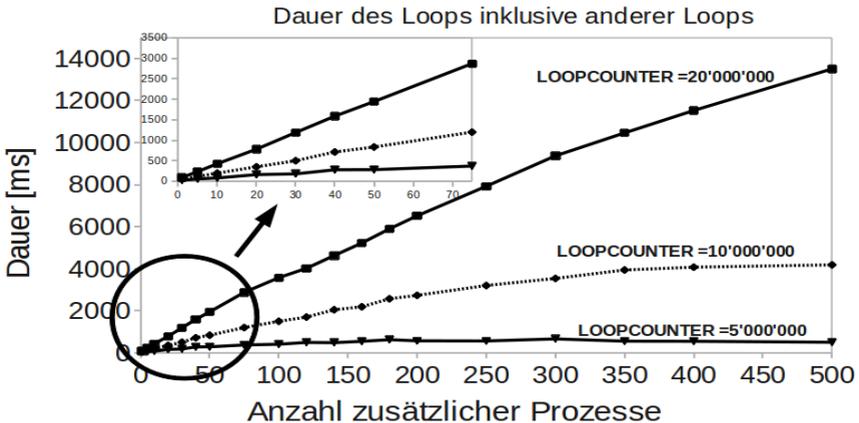
Um nun einen Prozess gegenüber einem anderen zu bevorzugen bietet POSIX das nice-Kommando an. Mit `nice programname` kann für ein Programm die Priorität reduziert werden und damit erhalten andere Programme, relativ gesehen, eine höhere Priorität. Man ist quasi nett (nice) zu den anderen Programmen. Eine andere und bessere Möglichkeit, die Priorität eines Programms zu verändern, ist die Verwendung der `nice(int inc)` Funktion aus dem `<unistd.h>` header-File. Hiermit kann die Priorität durch eine Reduzierung des nice-Wertes erhöht werden. Diese Reduzierung ist allerdings nur dem root-user vorenthalten (normale User können nur durch einen positiven inc-Wert die Priorität reduzieren, was einer Erhöhung des nice-Wertes entspricht). Nice-Werte bei Linuxsystemen liegen normalerweise bei -20 (höchste Priorität) und 19 (niedrigste Priorität). Für unser obiges Programm bedeutet die Erhöhung des nice-Wertes mit dem Befehl `nice(-20)`; eine Maximierung der Priorität. Zwar modifiziert diese kleine Änderung nicht die absolute Ausführungszeit des Loops allerdings wird plötzlich das Verhältnis von Prozesszeit und Realzeit bis auf wenige % gleich 1. Anstelle eines Faktors von 1.4 wie in der obigen Abbildung ergeben sich Werte von maximal 1.01. Unser Prozess besitzt nun also eine höhere Priorität als die Hintergrundprozesse, die vorhin unseren Loop ausgebremst haben und somit entspricht die Programmlaufzeit bei allen Messungen der Loop-Zeit. Auch zur Analyse der Hintergrundprozesse bietet Linux viele Tools. Hier sei nur das wohl bekannte Kommando `ps ax` erwähnt, bei dem Prozesse markiert sind, die einen Nicewert von <0 besitzen. Auch top besitzt eine Spalte die den nice-Wert angibt. Eine Analyse ergab, dass bei uns der Prozess udev mit einem Nicewert von -4 der obige Hintergrundprozess war, der bei den 2 Messungen die Realtime vergösserte. „udev“ ist ein Daemon zur Verwaltung des device-Systems von Linux.

### 4 Ein interessanteres Beispiel



Das bisher dargestellte Beispiel ist nicht besonders interessant, da nur ein Prozess die Ressourcen des Systems benötigt, die diesem Prozess natürlich auch fast immer vollständig zur Verfügung gestellt werden kann. Nur ganz selten werden andere, im Hinter-

grund blockierte Prozesse aktiv, die dann auch einige Ressourcen benötigen. Was passiert nun, wenn weitere Prozesse gestartet werden, die ebenfalls Ressourcen benötigen. Zur Erzeugung von weiteren Prozessen wird die Systemfunktion fork() mehrfach aufgerufen, die dann jeweils einen neuen Prozess kreiert, der ebenfalls in einem Loop eine Variable hochzählt. Das Programmkonstrukt ist im folgenden Bild dargestellt. Der 'Hauptprozess' misst wie oben die Zeit eines Loops, der eine Variable bis auf LOOPCOUNTER zählt. Ist dieser loop durchlaufen, so werden die 'Stress-Prozesse' beendet (durch wiederholte Aufrufe von kill) und die Dauer des Loops wird berechnet. In der unteren Abbildung sind die Ergebnisse für unterschiedliche LOOPCOUNTER-Werte angegeben: 20Mio, 10 Mio und 5 Mio. Für eine geringe Anzahl von 'Stressprozessen' sind die Realzeiten linear mit der Anzahl zusätzlicher Prozesse (kleines Bild), bei höhere Anzahl flachen die Kurven ab und die Zeiten ändern nicht mehr mit steigender Anzahl von Prozesse. Der Grund für dieses Verhalten ist erst auf den zweiten Blick klar: Die Dauer des Programms ist einige Sekunden, d.h. das Programm wird nur einige 100 mal preemptiv unterbrochen. In dieser Unterbruchzeit werden neue Prozesse erzeugt die dann jeweils ihren unendlichen loop abgearbeiten. Der Scheduler muss allerdings immer wieder auch unseren Messprozess aufrufen und hat deshalb bei einer grossen Anzahl von zusätzlichen Prozessen gar keine Zeit, alle verlangten Prozesse zu erzeugen. Dementsprechend ist unser Messloop auch schon zu Ende, bevor alle neuen Prozesse erzeugt sind und deshalb spielt auch die Zahl, wieviele Prozesse noch erzeugt werden müssten ab einer bestimmten Grenze keine Rolle mehr. Die Prozesszeiten sind mit 17.8 ms (5 Mio), 35.6 ms (10 Mio) und 71.2 ms (20Mio) ebenfalls linear und weitgehend unabhängig von der Anzahl weiterer Prozesse.



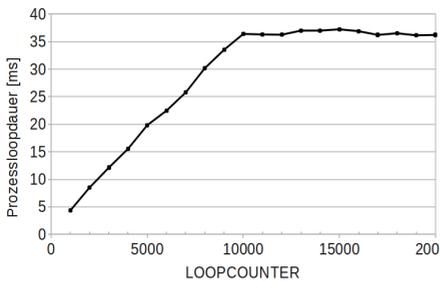
Um für die weiteren Untersuchungen nun noch interessantere Bedingungen zu erzeugen, wurde unser Messloop künstlich verlängert, in dem im loop noch auf verschiedene Seiten im Speicher zugegriffen wurde. Dazu wurde der Loop wie angegeben erweitert: Zuerst wird ein grosser Speicherbereich alloziert, hier 100 MB (`sizeof(struct stru)`), danach wird bei jedem Durchlauf des loops genau ein Byte pro Page (Pagegrösse 4kB) des Speichers geändert. Es müssen also eine grosse Anzahl (genau `NUM_PAGES`, bei uns 10'000) von Pages in den Speicher geladen und gehalten werden. Hiermit kann die Dauer von Speicherzugriffen inklusive dem Laden untersucht werden. Die Dauer des Messloops steigt nun auch ohne zusätzliche Prozesse auf ca. 400ms bei einem `LOOPCOUNTER` von 10 Millionen. Diese grosse Verlängerung der Loopdauer benötigt eine Erklärung:

```

...
ptr = malloc(sizeof(struct stru));
clock_gettime(CLOCK_REALTIME, &rt1);
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts1);
j=0;
while(1){
    zaehler++;
    j++;
    if(j>=NUM_PAGES){j=0;}
    ptr->arr[4096*j] = j;
    if(zaehler>LOOPCOUNTER){
        clock_gettime(CLOCK_REALTIME, &rt2);
        .....
    }
}

```

Der Loop selber besitzt nur einen Speicherzugriff mehr als im oberen Beispiel, es ist also erstaunlich, dass der Loop über 10 Mal länger dauert. Allerdings werden diese Zugriffe in verschiedenen Pages gemacht. Zuerst einmal müssen diese Pages erzeugt werden. Zwar wurden diese Pages bereits vor Starten der Messung im `malloc(...)` reserviert, angelegt werden sie allerdings erst beim ersten Zugriff darauf



(dieses Verhalten wird in einem etwas anderen Kontext als 'Copy on Write' bezeichnet). Dies bedeutet also, dass die Performance einer Routine erhöht werden kann, wenn auf alle Pages, auf die diese Routine zugreift, bereits vorher mindestens einmal beschrieben oder gelesen wurde. Damit ist die Page angelegt und es kann sehr schnell auf sie zugegriffen werden. In der

nebenstehenden Grafik ist dargestellt, wie die Prozesszeit variiert, wenn der Wert von `LOOPCOUNTER` ändert (ohne zusätzliche Stressprozesse). `NUM_PAGES` ist auf 10'000 gesetzt. Es ist deutlich ersichtlich, dass nur die ersten 10'000 Zugriffe auf den Speicher messbare Zeit benötigen, alle weiteren Zugriffe sind sehr schnell und steigen nur so langsam an, dass sie im Rauschen der Messung untergehen. Eine weitere interessante Messung ist die Variation von `NUM_PAGES` bei gleichbleibendem `LOOPCOUNTER`. Hier zeigt sich, dass sich bei wenigen Pages (bis 10) die Zeit nicht gegenüber der ersten Messung aus Kapitel 2 verändert, bei einer grösseren Anzahl von Pages steigt die Prozesszeit schrittweise an. Bei grosser Zahl (ab 1000) bleibt die Zeit quasi konstant bei ca. 420ms. Es scheint also, der Prozessors ständig Pages in den Cache lädt, die Pagetables aktualisieren und den Translation Lookaside Buffer (TLB) anpassen muss. Bei 32kB L1-Daten-Cache und

2MB L2-Cache des verwendeten Prozessors sind die gemessenen Zahlen zwar in der richtigen Größenordnung, können aber sicherlich nicht alles erklären.

Natürlich ist das in diesem Kapitel beschriebene Beispiel etwas künstlich und würde wohl so nicht in der Realität vorkommen. Allerdings erlauben es gerade solche Beispiele, das Verhalten des Linuxkernels zu studieren und besser zu verstehen. Ist dieses Verständnis erst vorhanden, ist es auch einfach eine reale Applikation zu optimieren.

## 5 Ändern der Scheduling-Policy

Linux bietet verschiedene Möglichkeiten, die Art und Weise, wie bei einem Kontextwechsel ein neuer Prozess ausgewählt wird, zu beeinflussen. Dies wird durch sogenannte Scheduling-Policies vorgegeben. Normalerweise laufen Prozesse mit einer Policy namens `SCHED_NORMAL` (definiert im Header-File `sched.h`). In aktuellen Linuxkernelversionen ist dieser als „completely fair scheduler“ implementiert, also als Scheduler, der jedem User die gleiche Wahrscheinlichkeit zuteilt, dass einer seiner Prozesse ausgewählt wird. Zwar werden Prozesse mit höherer Priorität bevorzugt, aber auch Prozesse mit sehr niedriger Priorität kommen ab und zu zum Zuge.

Für einzelne Prozesse können nun auch andere Policies ausgewählt werden. Bei `SCHED_FIFO` (`FIRST_IN-FIRST-OUT`) wird jeweils der Prozess mit der höchsten Priorität ausgewählt und dieser Prozess erhält solange die Prozessorressourcen, bis er entweder in seinen Blocking-State übergeht, oder ein Prozess mit höherer Priorität aktiv wird. Diese Policy ist für Soft-Realtime Prozesse vorgesehen. Allerdings kann auch bei `SCHED_FIFO` nicht wirklich von Realtime gesprochen werden, da aufgrund der komplexen Struktur der Speicherzugriffe (virtual Memory), der nichtdeterministischen Cache-Struktur und der Pipeline-Tiefe heutiger Prozessoren kein wirkliches Realtime-Verhalten möglich ist. Es gibt allerdings Zusätze für Linuxsysteme (z.B. [www.xenomai.org](http://www.xenomai.org), [www.RTLinux.org](http://www.RTLinux.org), ...), die eine harte Realtimefähigkeit mit Linux ermöglichen.

Im Gegensatz zu `SCHED_FIFO` werden bei `SCHED_RR` (Round-Robin) Prozesse mit gleicher Priorität hintereinander ausgewählt. Prozesse mit niedrigerer Priorität werden allerdings erst wieder ausgewählt, wenn höher priorisierte Prozesse blockieren oder beendet sind.

Als Interface zum Ändern der Policy dienen die Routinen `sched_getscheduler(...)` and `sched_setscheduler(...)`. Für weitere Informationen und die vielseitigen Möglichkeiten der Parametrisierung sei auf die jeweiligen Man-pages oder auf [1] verwiesen. Weiterhin sei auf [2] verwiesen, in dem noch modernere Methoden zur Kontrolle der Realtime-Eigenschaften von Prozessen beschrieben sind.

## 6 Halten von Daten im Speicher

Wenn von einem Programm zu viel Speicher alloziert wird, muss ein Teil auf ein anderes Medium, z.B. eine Festplatte ausgelagert werden. Dies wird als Swapping bezeichnet. Das Laden eines geswappten Speicherbereichs dauert natürlich wesentlich länger, da Festplattenzugriffe sehr langsam sind. Möchte man nun erreichen, dass der Speicher von einem bestimmten Prozess immer im Speicher bleibt und nicht geswappt wird, so bietet Linux die `mlock(...)` und `mlockall(...)`-Routinen aus dem `<sys/mman.h>` Header-File. Es ist leicht einzusehen, dass bei speicherintensiven Anwendungen durch diese Routinen die Performance für ausgewählte Prozesse positiv beeinflusst werden kann.

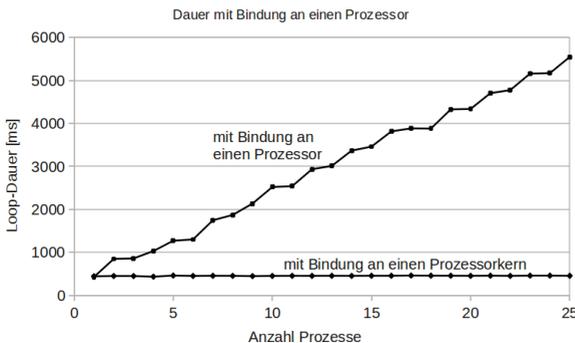
## 7 Binden eines Prozesses an einen Prozessor

Bei Mehrprozessorsystemen wie sie heute in Servern, Desktop-Computern und Laptops üblich sind, gibt es noch eine weitere sehr gute Möglichkeit, einzelne Prozesse zu optimieren. Es ist nämlich möglich, einem einzelnen Prozess genau einem Prozessor zuzuordnen. Wenn alle anderen Prozesse einem anderen Prozessor zugeordnet werden, so hat unser „wichtiger“ Prozess seinen Prozessor alleine.

Um dies zu erreichen dient die Routine

```
int sched_setaffinity(pid_t pid, size_t setsize, const cpu_set_t *set);
```

mit der angegeben werden kann, welcher Prozess (pid) an einen bestimmten Prozessor (bestimmt durch \*set) gebunden werden soll. 'set' wird durch die Makros CPU\_ZERO, CPU\_SET und CPU\_CLR bestimmt. Für weitere Informationen zu diesen Interfaces sei auf die entsprechenden Man-Pages verwiesen. Die folgende Abbildung zeigt Messungen an unserem Problem. Hier wurde wieder unser Mess-Loop (10 Millionen Zugriffe auf 10'000 Pages im Speicher) mit einer unterschiedlichen Zahl von zusätzlichen Prozessen 'gestresst' und die Zeit gemessen, die er für das Durchlaufen des loops benötigt. Ohne Bindung an einen Prozessor gibt es einen mehr oder weniger linearen Zusammenhang zwischen Anzahl



zusätzlichen Prozessen 'gestresst' und die Zeit gemessen, die er für das Durchlaufen des loops benötigt. Ohne Bindung an einen Prozessor gibt es einen mehr oder weniger linearen Zusammenhang zwischen Anzahl

Prozessen und Loopdauer, mit Bindung an einen Prozessor und Bindung der Stress-Prozesse an den anderen Prozessor, wird der loop durchlaufen, als ob keine anderen Prozesse vorhanden wären. Interessant an der nebenstehenden Abbildung ist auch das paarweise Auftreten der Prozesse zum Verlängern der Loopdauer. Natürlich wird bei unserem Dualcore-Prozessor nur bei jedem 2-ten zusätzlichen Prozess der Messloop mehr belastet, was zu einer Verlängerung der Zeit führt.

## 8 Schlussfolgerungen und Ausblick

Leider konnten in dieser kurzen Abhandlung viele interessante Gesichtspunkte nur kurz angesprochen werden. Es gibt viele Möglichkeiten ein Linuxsystem zu beeinflussen und ein tieferes Verständnis der dynamischen Eigenschaften dieses Systems zu gewinnen. Richtig eingesetzt können diese Erkenntnisse die Performance von Applikationen massiv erhöhen. Interessant wird es besonders bei Multiprozessorsystemen. Linux unterstützt symmetric Multiprocessing also ein System, bei dem Prozesse symmetrisch auf die Prozessoren verteilt werden. Dies ist in der Desktop-Welt sicher sinnvoll, da diese Geräte generisch sind und viele verschiedene Aufgaben erledigen müssen. In der embedded Welt, in der die Aufgaben bereits oft schon während der Designphase der Software klar sind, ist dies sicherlich anders, da dort auch asymmetrische Architekturen vorstellbar, vielleicht sogar wünschenswert sind. Embedded ARM-Prozessoren z.B. enthalten häufig noch Hardwarekomponenten, die DSP- (Digital Signal Processing) und/oder SIMD (Single Instruction, Multiple Data)-Funktionalitäten bieten, welche heute noch, mangels vernünftiger Einbindung in den Linuxkernel kaum verwendet werden. Hier werden sicherlich in den nächsten Jahren sehr viele und interessante Erweiterungen entstehen, die die Performance des GNU/Linuxsystems einen weiteren Performanceboost bescheren wird.

### Literatur

- [1] Robert Love: LINUX System Programming, O'REILLY 2007, ISBN: 978-0-596-00985-8
- [2] Jérôme Delamarque: Controle des processus avec les CGROUPS, GNU/Linux-Magazine France No 141, pp 8-15.
- [3] Daniel P. Bovet, Marco Cesati: Understanding the Linux Kernel, O'Reilly 2006, ISBN: 978-8184040838
- [4] Linux Performance and Tuning Guidelines, IBM-Redbooks 2008, <http://www.redbooks.ibm.com/abstracts/redp4285.html>



# Starke Zweifaktorauthentisierung mit LinOTP - modular, skalierbar, frei

Cornelius Kölbel

cornelius.koelbel@sexperts.de

<http://linotp.org>

LinOTP ist eine innovative und flexible Lösung zur Zweifaktorauthentisierung. Dank seiner modularen Architektur ist LinOTP herstellerunabhängig und unterstützt beliebige Token, Authentisierungsprotokolle und Datenbanken. LinOTP verwendet bevorzugt Einmalpasswörter, um die Sicherheit in Anmeldeverfahren zu erhöhen. Hier soll tiefer auf die Struktur von LinOTP eingegangen werden, um die flexible Einsetzbarkeit für verschiedenste Szenarien besser bewerten zu können.

## 1 Zweifaktor-Authentisierung

Das Bewusstsein, dass die Anmeldung an Computern oder Online-Diensten einzig und alleine mit einem Passwort nicht ausreichend sicher sein kann, hat sich aufgrund von regelmäßig auftretenden Vorfällen auch bei den Endanwendern bereits gebildet. Die Idee, sich neben dem *Wissen* des Passworts mit einem zweiten Faktor *Besitz* auszuweisen, hat selbst in der Neuzeit schon eine lange Geschichte hinter sich.

1968 bereits identifizierte HAL in *Odyssee im Weltraum* seine Mitreisenden über Stimmerkennung. 1971 wurden von James Bond in *Diamantenfieber* Fingerabdrücke verwendet. 1982 sollten von den *Bladerunnern* die Replikaten anhand eines Retinascans identifiziert werden und im gleichen Jahr musste sich Captain Kirk im *Zorn des Khans* ebenfalls einem Retinascan unterziehen, um Zugriff auf die Genesis Projektdaten zu erhalten. 4 Jahre später — 1986 — wurde der zweite Faktor in Form eines zusätzlichen Gerätes mit dem RSA SecurID<sup>®</sup> Token auch in der realen Welt alltagstauglich[1].

Danach hat sich der *Besitz* als zweiter Faktor gegenüber dem *Besitz eines biometrischen Merkmals* weiter verbreitet, so dass der *Besitz* heute im Wesentlichen in Form von Smartcards und OTP-Token (One Time Password) realisiert wird. Während bei Smartcards asymmetrische Kryptographie (i.d.R. RSA) zum Einsatz kommt, werden im Fall von OTP-Token symmetrische Algorithmen genutzt. Beide Möglichkeiten haben Ihre Vor- und Nachteile[2]. Aufgrund der weniger komplexen Infrastruktur und der geringeren Betriebsaufwände erfreuen sich OTP-Token gegenüber Smartcards nun wieder einer wachsenden Beliebtheit.

## 2 Verschiedene OTP-Algorithmen

Die heute gängigen OTP-Algorithmen verwenden einen geheimen symmetrischen Schlüssel und einen Zähler. Der Schlüssel (oder auch *Seed*) ist sowohl bei dem Benutzer als auch bei dem authentifizierenden Backend vorhanden. Herausforderungen bestehen darin, diesen geheimen Schlüssel initial zwischen dem Benutzer/Token und dem Backend auszutauschen und zu verhindern, dass Dritte an diesen Schlüssel gelangen. Der Zähler (auch *moving factor* genannt) kann ereignisgesteuert oder zeitgesteuert sein. Dieser stellt sicher, dass sich das generierte Einmal-Passwort verändert und eben nur einmal verwendet werden kann. Abstrakt lässt sich die Berechnung der verschiedenen OTP-Algorithmen wie folgt darstellen:

$$OTP_{value} = otpAlgorithm(Seed, movingFactor) \quad (1)$$

Der Zähler lässt sich sowohl aus einem Ereignis (i.d.R. der Tastendruck auf dem Token) sowie aus der Zeit erzeugen. Hierzu muss in dem Token eine Uhr laufen, die synchron mit dem Backend sein muss. Meist wird die POSIX Zeit in entsprechende Zeitscheiben von 30 oder 60 Sekunden geteilt, so dass sich am 13.01.2001 11:27:30 mit der POSIX Zeit 1326450450 folgender Zähler ergeben kann:

$$movingFactor_{time} = 1326450450 / 30 = 44215015 \quad (2)$$

### 2.1 Betrachtung des Zählers

**Ereignisbasierte Zähler** haben kein Problem mit Zeitdrift bzw. weniger Probleme mit dem Batterieverbrauch, da die Batterie nur bei Anfordern eines OTP-Wertes verwendet wird. Der OTP-Wert wird erst ungültig, wenn er zur Authentisierung genutzt wurde. Somit lassen sich *TAN-Listen* erzeugen. Das Backend muss ein Zähler-Fenster verwalten. Ein ereignisbasierter Token lässt sich nicht leicht mit mehreren Backends betreiben.

**Zeitbasierte Zähler** haben ein größeres Problem mit Zeitdrift oder Batterieverbrauch, da die Uhr immer tickt. Das Backend muss ein Zeitdrift-Fenster verwalten, um damit umgehen zu können, wenn sich die Uhrzeit im Token über die Zeit stark verändert. Die Erzeugung von *TAN-Listen* ist nicht möglich. Eine Anmeldung ist nur alle 30 bzw. 60 Sekunden möglich, da man dann erst einen neuen OTP-Wert erhält. Ein zeitbasierter Token kann gleichzeitig mit mehreren Backends betrieben werden.

### 2.2 Proprietäre Algorithmen

Der Wegbereiter RSA[3] verwendet in seinen zeitbasierten SecurID<sup>®</sup> Token weiterhin Algorithmen, die der Öffentlichkeit nicht bekannt sind. Der deutsche Hersteller

Kobil[4] verwendet ebenfalls in seinen zeitbasierten SecOVID Token einen proprietären Algorithmus. VASCO[5] ist der Hersteller, der seine DigiPass Token bei Online-Diensten wie ebay, paypal und blizzard untergebracht hat. Doch auch hier benutzt die Standard-Ausführung der Token einen nicht öffentlichen zeitbasierten Algorithmus.

## 2.3 Freie Algorithmen

Die *Initiative for Open Authentication*[10] treibt die Standardisierung von Authentisierungsalgorithmen voran. Zur Zeit dreht es sich dort speziell um OTP. In den RFCs 4226, 6238 und 6287 sind verschiedene freie Algorithmen spezifiziert.

### 2.3.1 HOTP — RFC4226

Dies ist ein ereignisbasierter Algorithmus, der den Message Authentication Algorithmus HMAC-SHA-1[7]<sup>1</sup> nutzt:

$$\begin{aligned} OTP_{value} &= HOTP(Seed, movingFactor) \\ &= Truncate(HmacSHA1(Seed, movingFactor)) \end{aligned} \quad (3)$$

Der so in dem Token berechnete und an das Backend zur Authentifizierung gesendete OTP-Wert wird versucht im Backend ebenso auszurechnen. Dazu nutzt auch das Backend den Seed des Tokens und den letzten bekannten Zählerstand. Da auf dem ereignisbasierten Token mehrere OTP-Werte erzeugt worden sein könnten, ohne dass sie an das Backend geschickt wurden, kann es sein, dass der Zähler schon um mehr als eins fortgeschritten ist. Daher kann das Backend im Rahmen des vorher erwähnten Zähler-Fensters weitere OTP-Werte zum Vergleich berechnen.

Eine einfache Implementierung in python sieht wie folgt aus.

```
1 import hmac, struct, binascii, hashlib
2
3 key_hex = "11223344556677889900AABBCCDDEEFF11223344"
4
5 def truncate(digest, digits=6):
6     '''truncates to 6 or 8 digits'''
7     offset = ord(digest[-1:]) & 0x0f
8     binary = (ord(digest[offset + 0]) & 0x7f) << 24
9     binary |= (ord(digest[offset + 1]) & 0xff) << 16
10    binary |= (ord(digest[offset + 2]) & 0xff) << 8
11    binary |= (ord(digest[offset + 3]) & 0xff)
12    return binary % (10 ** digits)
13
```

<sup>1</sup>Es existieren Varianten, die nicht einen 160 Bit langen Schlüssel und SHA1 sondern einen 256 Bit langen Schlüssel und SHA2-256 verwenden.

```

14 def hotp(key_hex, counter, digits=6):
15     key = binascii.unhexlify(key_hex)
16     digest = hmac.new(key, struct.pack(">Q", counter),
17         hashlib.sha1)
18     otp = str(truncate(str(digest.digest())))
19     """ fill in the leading zeros """
20     return (digits - len(otp))*"0"+otp
21
22 for i in [1,2,3,4]:
23     print hotp(key_hex, i)

```

Hier werden zu einem gegebenem Seed (`key_hex`) die ersten vier OTP-Werte in einer sechsstelligen Variante berechnet.

### 2.3.2 TOTP — RFC6238

Der in RFC6238[8] definierte TOTP Algorithmus entspricht dem HOTP Algorithmus mit dem Unterschied, dass der Zähler nicht ereignisbasiert hochgezählt wird, sondern zeitbasiert aus der POSIX Zeit berechnet wird. Der RFC erlaubt Zeitschritte von 30 oder 60 Sekunden, so dass sich der OTP-Wert wie folgt berechnet:

$$OTP_{value} = HOTP \left( Seed, \frac{Time_{posix}}{30} \right) \quad (4)$$

```

1 import time
2 totp = hotp(key_hex, int( time.time() /30))

```

### 2.3.3 OCRA — RFC6287

Der HOTP und der TOTP Algorithmus haben den Vorteil, dass der Token völlig autark funktionieren kann. Jedoch sind die Einsatzszenarien rein auf die Authentisierung beschränkt. Der Token-Besitzer authentisiert sich gegenüber dem Backend mit einem Zähler, den der Token selber bestimmt.

Der OATH Challenge-Response Algorithm[9] bietet daneben weitere Möglichkeiten verschiedene Eingangsdaten und Algorithmen zu verwenden. Einfach gesagt wird hier der Zähler durch eine Challenge, die das Backend an den Benutzer/den Token sendet, ersetzt.

$$\begin{aligned}
 OTP_{value} &= OCRA(Seed, DataInput) \\
 &\approx HOTP'(Seed, Challenge)
 \end{aligned} \quad (5)$$

### 2.3.4 mOTP

Der mOTP[11] Algorithmus ist ein freier Algorithmus, der sich einer festen Community erfreut, der aber in keinem RFC spezifiziert ist. Das Projekt definiert einen Algorithmus zur Verwendung als Applikation auf Smartphones und alten Featurephones. Der Algorithmus ist zeitbasiert und stellt sich wie folgt dar:

$$OTP_{value} = md5(Time_{posix} + pin + seed) \quad (6)$$

Es fehlt eine Security-Betrachtung zur Abwägung der Nutzung des MD5 Algorithmus gegenüber der SHA Algorithmen-Familie bei HOTP und TOTP. Dieser Algorithmus erlaubt es aber einen Soft-Token als Applikation auf sehr alten Telefonen zu betreiben.

## 3 Token Typen

**Soft-Token** sind Token, die als Software auf einem Computer oder auf einem Mobiltelefon/Smartphone laufen. Der Nachteil der Soft-Token ist, dass Sie sich leicht duplizieren lassen und dass manchmal der Austausch des Schlüssels zwischen Software und Backend aufwändig ist und einen Angriffsvektor zur Kompromittierung des Schlüssels bietet. Andererseits sind Soft-Token gerade wegen der Einfachheit in der Bedienung und aufgrund der Tatsache, dass ein bereits vorhandenes Gerät verwendet wird, sehr akzeptiert. Typische Vertreter sind der mOTP-Token[?] und der Google Authenticator[12].

**SMS-Token** senden den OTP-Wert per SMS an ein Mobiltelefon. Der Vorteil ist, dass hier keine Hardware ausgeteilt werden muss und auf den Telefonen auch keine App installiert werden braucht. Oftmals erweist sich die Fragestellung, wie der SMS Versand ausgelöst werden soll, als problematisch. Das Sicherheitsniveau wird durch die nicht vorhandene Security im GSM Netz bestimmt.

**Pre-seeded Hardware-Token** bilden das Gros der am Markt vertretenen Token. Dies sind kleine Schlüsselanhänger mit Display und einem Knopf, um entweder das Display bei zeitbasierten Token einzuschalten oder bei ereignisbasierten Token den Zähler zu erhöhen. Diese Token sind von verschiedenen Herstellern wie RSA, SafeNet, VASCO, Authenex, Feitian usw. erhältlich. Da mit dem HOTP Algorithmus ein leicht zu implementierender Algorithmus gegeben ist, werden gerade auf dem chinesischen Markt solche Token in großen Mengen von vielen Herstellern produziert. Diese Token bekommen in der Fabrik bereits einen Seed einprogrammiert, der mit einfachen Mitteln nicht mehr aus dem Token ausgelesen werden kann. Jedoch muss das Seed bei diesen Token in einer Datei mitgeliefert werden, was das Seed aus der Datei nun wieder leicht kompromittierbar macht.

**Seed-bare Hardware-Token** sind Token, die ohne Seed aus der Fabrik kommen. Als Anwender kann man selber auf diese Token ein Seed aufbringen. Heute ermöglichen dies erst ein paar wenige Token. Dem eTokenNG OTP (SafeNet) kann

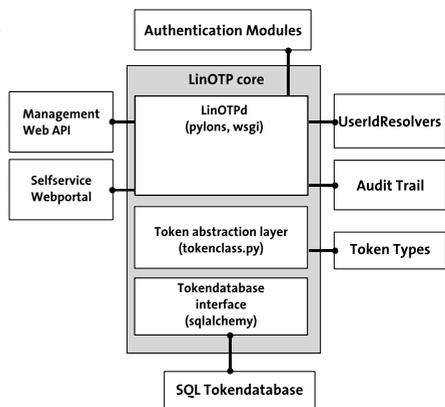
über einen USB-Anschluss ein neuer geheimer Schlüssel aufgebracht werden, für den eTokenPASS (SafeNet) gibt es ein Programmiergerät, um einen geheimen Schlüssel aufzubringen. Der Yubikey (yubico), der nicht mit Display sondern als Tastatur per USB am Computer angeschlossen wird, kann ebenfalls über USB einen neuen geheimen Schlüssel bekommen.

Es kann verschiedene Gründe geben, wieso unterschiedliche Token-Typen in verschiedenen Szenarien verwendet werden. Oftmals ist es eine Abwägung aus Kosten und Sicherheitsbedürfnis zwischen einem Soft-Token und einem seed-baren Hardware-Token zu wählen. Einen blinden Mitarbeiter wird man nicht mit einem OTP-Token ausstatten, der auf einem kleinen Display das Einmalpasswort anzeigt. Hier empfiehlt sich die Nutzung eines Yubikeys, der ohne Display, nur nach Tastendruck den OTP-Wert in die aktuelle Eingabezeile hineinschreibt.

Im Enterprise-Umfeld sehen wir immer mehr Firmenzusammenschlüsse. Hier ist es oft nicht Wunsch sondern unangenehmer Nebeneffekt, im neuen, zusammengewachsenen Unternehmen plötzlich viele verschiedene Token-Typen vorzufinden. Statt neue Token an alle Benutzer auszurollen ist es wünschenswert, die bereits ausgerollten in einem einheitlichen System weiterzuverwenden.

## 4 LinOTP

LinOTP[13] versucht als modulares Authentisierungs-Backend diese heterogenen Szenarien zu bedienen. LinOTP ist in python geschrieben. UserIdResolver, Audit Trails und Token-Typen werden als vererbte Klassen der jeweiligen Basis-klassen<sup>a</sup> eingebunden. LinOTP bedient sich des Pylons Webframeworks<sup>b</sup>. Die Protokolle zur Authentisierung und zum Management sind einfache Webrequests, die leicht parsebares JSON zurückliefern. Die Tokendaten werden in einer SQL-Datenbank gehalten. Durch Nutzung von SQLAlchemy ist das Datenbank-Backend leicht austauschbar.



<sup>a</sup>Die genauen Klassenbeschreibungen finden sich *Modularer Aufbau von LinOTP 2* auch auf der Projektwebseite[?].

<sup>b</sup><http://www.pylonsproject.org/>

## 4.1 Klassendefinitionen

### 4.1.1 UserIdResolver

Das Konzept der UserIdResolver, von denen mehrere zu Realms zusammengefasst werden können, ist in LinOTP ein mächtiges Werkzeug, um Benutzer faktisch in beliebigen Datenquellen zu finden und ihnen mehrere Token zuordnen zu können. Dabei wird immer nur lesend auf die Datenquellen zugegriffen. Der UserIdResolver ist dafür zuständig, den Benutzer in einer Datenquelle zu finden und ihn in LinOTP mit einer eindeutigen ID zu adressieren. Deswegen definiert auch jeder UserIdResolver selbst, wie ein Benutzer in dieser Datenquelle eindeutig identifiziert werden kann.

Es existiert ein LDAPResolver für beliebige LDAP-Server (Active Directory, Novell eDirectory, OpenLDAP), ein SQLResolver für verschiedenste SQL-Datenbanken und ein Resolver für einfache Dateien, die die Form einer \*NIX passwd haben können. Dabei implementieren die UserIdResolver u.a. die Methoden:

**getResolverId** gibt für einen gegebenen Login-Namen die eindeutige Id eines Benutzerobjektes zurück.

**getUserId** gibt für eine eindeutige Benutzerobjekt-Id den Login-Namen zurück.

**getUserInfo** gibt für eine eindeutige Benutzerobjekt-Id detaillierte Benutzerinformationen (Bspw. Vorname, Nachname, Email, Telefonnummer...) aus.

**getUserList** liefert eine vollständige Liste der in diesem UserIdResolver gefundenen Benutzer. Ein Searchpattern kann angewendet werden.

**loadConfig** lädt die Konfiguration des UserIdResolvers. Die Konfiguration kann im Falle des LDAPResolvers enthalten, in welcher OU Benutzer gesucht werden sollen oder welches das Attribut ist, das den Login-Namen enthält.

**checkPass** überprüft das Passwort des Benutzers in der jeweiligen Datenquelle. Diese Funktion wird zum Login in das Selfservice-Portal oder optional als statischer zweiter Faktor zusammen mit dem OTP-Wert verwendet. Da die Passwort-Überprüfung im UserIdResolver implementiert ist, kann die Authentifizierung des Benutzers völlig transparent erfolgen, egal ob dieser sich in einem LDAP-Verzeichnis, in einer SQL-Datenbank oder in /etc/passwd liegt.

Theoretisch können leicht weitere Module (Klassen) für andere Benutzerquellen da-  
zuentwickelt werden. Bisher konnten aber alle Fälle mit den drei existierenden Resol-  
vern abgedeckt werden.

### 4.1.2 Token-Typen

Alle Token sind in einer Token-Tabelle gespeichert. Diese enthält für jeden Token einen Eintrag, mit der Referenz zu dem zugeordneten Benutzer (UserIdResolver, Benutzerobjekt-Id). Sensible Daten wie das Seed und die OTP PIN sind verschlüsselt bzw. gehasht abgelegt. Ein Object-Relational-Mapping (ORM) bildet die Datenbank-Tabelle auf die Tokenklasse ab. Die Token-Basisklasse stellt viele verschiedene Basisfunktionalitäten sicher, so dass oftmals nur wenige Methoden überschrieben werden müssen, um einen neuen Token-Typ aufzunehmen:

**checkOtp** ist die zentrale Methode, die den OTP-Wert mit dem entsprechenden Algorithmus des jeweiligen Tokens überprüft.

**splitPinPass** teilt das eingegebene, zu überprüfende Passwort in den festen Passwortbestandteil und den variablen — den OTP-Wert.

**checkPin** überprüft den festen Passwortbestandteil des eingegebenen Passwortes.

Auf diese Art und Weise wurden bisher folgende Typen implementiert:

**HmacTokenClass** bildet die HOTP Token ab. Dies können beliebigen Hardware-Token oder auch Soft-Token wie der Google Authenticator sein.

**TimeHmacTokenClass** bildet die TOTP Token ab. Auch diese können entweder Hard- oder Soft-Token sein.

**MotpTokenClass** bildet die mOTP Token ab.

**SpassTokenClass** ist ein *Simple Pass Token*. Dieser spezielle Token benutzt keinen OTP-Wert sondern zieht ausschließlich die OTP-PIN zur Authentifizierung heran.

**PasswordTokenClass** wird für den Workflow eines verlorenen Tokens benutzt. Diese Tokenklasse benutzt als OTP-Wert ein langes, statisches Passwort, so dass die Authentisierung mit einer OTP-PIN und dem langen, statischen Passwort erfolgen muss.

**SmsTokenClass** sendet OTP-Werte per SMS. Hier können verschiedene Module definiert werden, wie die SMS versendet werden soll (über ein SMS-Gateway oder ein angeschlossenes GSM-Gerät).

**RadiusTokenClass** definiert Token, die die Authentisierungsanfrage an einen RADIUS-Server weiterleiten können.

**RemoteTokenClass** definiert Token, die die Authentisierungsanfrage an einen anderen LinOTP-Server weiterleiten können.

**TagespasswortTokenClass** definiert Token, die ihr Passwort nur einmal am Tag ändern.

**VascoTokenClass** bildet die Vasco DigiPass Token ab. Dies ist der erste unterstützte proprietäre Algorithmus. Hierzu wird eine Library von Vasco eingebunden. Alle Daten werden aber nach wie vor in der LinOTP eigenen Token-Datenbank verschlüsselt abgelegt.

Dies zeigt, dass sich die Liste der unterstützten Token-Typen schnell und leicht erweitern lässt. Mit dem Vasco-Token wiederum wurde demonstriert, dass das Konzept von LinOTP bei Bedarf auch für proprietäre Token bestens verwendet werden kann und alle verschiedenen Token-Typen gleichzeitig und nahtlos genutzt werden können.

### 4.1.3 Audit Trail

LinOTP bietet eine oft am Markt gesuchte und selten gefundene Möglichkeit alle Aktivitäten zu loggen und auditieren. Hierüber lassen sich bei Bedarf auch PCI-DSS[14] Anforderung sicherstellen.

LinOTP erzeugt Audit-Informationen, die an das konfigurierte Audit-Modul übergeben werden. Die Informationen, die erzeugt werden, reichen von den Aktivitäten eines Administrators über die Tätigkeiten eines Support Mitarbeiters bis hin zu den Aktivitäten des Benutzers. Damit lassen sich auch kompletten Lebenszyklen eines Tokens nachvollziehen.

number	date	signature	action	success	serial	user	administrate	action_detail
12530	2011-12-12 22:36:29	OK	admin/remove	1	LSGO1112051347534916		admin	tokennum = 14
12416	2011-12-08 14:19:28	OK	validate/check	0	LSGO1112051347534916	koelbel		wrong otp value
12415	2011-12-08 11:40:59	OK	validate/check	0	LSGO1112051347534916	koelbel		wrong otp value
12410	2011-12-08 10:19:57	OK	validate/check	1	LSGO1112051347534916	koelbel		
12408	2011-12-08 10:18:11	OK	admin/unassign	1	LSGO1112051347534916		admin	
12407	2011-12-08 10:17:55	OK	validate/check	1	LSGO1112051347534916	koelbel		
12390	2011-12-08 10:16:12	OK	admin/enable	1	LSGO1112051347534916		admin	
12383	2011-12-08 10:04:55	OK	admin/tokenrealm	1	LSGO1112051347534916		admin	
12285	2011-12-08 10:02:23	OK	admin/unassign	1	LSGO1112051347534916		admin	
12227	2011-12-05 13:50:05	OK	validate/check	0	LSGO1112051347534916	koelbel		Token inactive
12225	2011-12-05 13:49:38	OK	admin/losttoken	0	LSGO1112051347534916		admin	from LSGO1112051347534
12217	2011-12-05 13:48:41	OK	validate/check	1	LSGO1112051347534916	koelbel		
12215	2011-12-05 13:48:16	OK	selfservice/usersetpin	1	LSGO1112051347534916	koelbel		
12214	2011-12-05 13:47:53	OK	selfservice/userwebprovision	1	LSGO1112051347534916	koelbel		tokennum = 98

*Lebenszyklus eines Tokens, der vom Benutzer selber ausgerollt wurde, der um 13:49 Uhr als verloren (action=losttoken) gemeldet wurde und schließlich am 12.12.2011 vom Administrator admin gelöscht wurde.*

Die Audit Trail Basisklasse sieht vor, dass alle Audit Einträge digital signiert werden können und bei der Ansicht des Audit Trails die Signatur der Einträge überprüft wird.

Zur Zeit existieren ein FileAudit und ein SQLAudit Modul. Im Wesentlichen implementieren diese Klassen die Funktionen **log** und **search**, um die log Ereignisse zu schreiben bzw. wieder auszugeben.

## 4.2 Web-API

Neben der Erweiterbarkeit von LinOTP über die leichte Definition neuer Klassen bzw. Module, strebt LinOTP auch in den Bereichen Management und Authentisierung größte Flexibilität an und will weder vorschreiben, wie ein Management Tool aussehen soll oder welches das bevorzugte Authentisierungsprotokoll wäre. Gleichzeitig soll LinOTP automatisierbar steuerbar sein.

Dazu wurde der Ansatz einer Web-API basierend auf dem pylons framework gewählt. Die genaue API-Beschreibung findet sich auf der Projektwebseite[13].

### 4.2.1 Authentisierung

Die Authentisierung erfolgt über einen Controller */validate*, der verschiedene Methoden (check, check\_s, simplecheck, samlcheck, smspin) bereitstellt.

<http://linotp/validate/check?user=benutzer&pass=pin228769>

Der oben angegebene GET Request liefern folgendes Ergebnis zurück:

```
1 {
2     "version": "LinOTP_2.4",
3     "jsonrpc": "2.0",
4     "result": {
5         "status": true,
6         "value": false
7     },
8     "id": 0
9 }
```

Hierbei zeigt der *status* die erfolgreiche Verarbeitung des Requests an und der *value* (false), dass die Authentisierung fehlgeschlagen ist.

Über diese einfache Web-API lassen sich Web-Applikationen schnell um die Authentisierung mit einem zweiten Faktor erweitern. Um Standard-Protokolle bedienen zu können, wurden außerdem diverse Plugins wie ein *pam\_linotp* für den PAM Stack und ein *rlm\_linotp* für den FreeRADIUS-Server entwickelt. So hat man eine klare modulare Trennung: Nicht der LinOTP-Server selber muss als RADIUS-Server fungieren sondern der bereits existierende FreeRADIUS-Server wird über das *rlm\_linotp* nun in die Lage versetzt über eine definierte, öffentlich bekannte API mit dem LinOTP-Server zu kommunizieren.

## 4.2.2 Management

Auch das Management wird über verschiedene Controller (admin, system, audit) abgebildet. Über den system-Controller werden systemweite Einstellungen im LinOTP konfiguriert. Den zentralen Controller bildet der admin-Controller, der über Methoden wie `init`, `assign`, `unassign`, `enable`, `disable`, `set`, `tokenrealm`, `userlist` usw. alle Token im System managebar macht. So kann über den URL-Aufruf von

```
https://linotp/admin/init?otpkey=...&type=HMAC&user=benutzer&serial=...
```

ein neuer Token erzeugt und dem *benutzer* zugewiesen werden<sup>2</sup>. Auf diese Art und Weise können alle Managementfunktionen aufgerufen und in bereits bestehende Applikationen leicht eingebunden werden.

Natürlich bietet LinOTP selber ein Management Webinterface sowie einen nativen GTK-Client und ein Command Line Interface. Doch auch hier gilt wieder, dass der Entwicklung eines eigenen Management Interfaces nichts im Wege steht.

## 5 Szenarien

LinOTP ist ein modulares sehr flexibles Authentisierungsbackend, das gut skaliert und sich damit in vielen Szenarien einsetzen lässt.

An wenig komplexen Szenarien wie der Authentisierung eines Benutzers an einer einzelnen Maschine erweist sich die Benutzung einfacher Tools als praktikabel. In dem Zusammenhang hat M. Lorenz[16] das OATH Tool[15] bereits vorgestellt. Sind mehrere Benutzer, die verwaltet und supportet werden müssen, oder mehrere Maschinen, an denen die Benutzer sich authentisieren, involviert, so empfiehlt sich der Einsatz eines zentralen Authentisierungsbackends wie LinOTP.

Verbreitete Einsatzszenarien sind die Authentisierung an VPNs für Remotezugriffe und Authentisierung an Terminalserver-Diensten mit Nutzung des RADIUS-Protokolls. Dies ist ein übliches Szenario sowohl für kleine Benutzergruppen, KMUs und große Konzerne. Im universitären Umfeld kann LinOTP durch flexibelste Anbindung von Benutzerquellen seine Stärken ausspielen. Mittels *pam\_radius* oder *pam\_linotp* lassen sich gezielt \*NIX Systeme mit einem zweiten Faktor absichern. LinOTP läuft auf Linux und kann daher auch bei Providern und auf gehosteten und virtualisierten System ressourcenschonend und erfolgreich eingesetzt werden.

## 6 Ausblick

Das Credo von LinOTP ist Offenheit, Flexibilität und Modularität. Daher geht die Reise weiter in Richtung Flexibilität des zweiten Faktors. LinOTP wird flexibler in der

---

<sup>2</sup>Der Einfachheit halber ist die Authentisierung und Session hier weggelassen

Unterstützung des zweiten Faktors. Weg von der reinen Bindung an Einmalpasswörter hin zu einer flexiblen Zweifaktor-Authentisierungslösung — sei der zweite Faktor ein Zertifikat auf eine Smartcard, biometrische Merkmale oder Dinge wie die ganz persönliche Unterschrift.

## Literatur

- [1] RSA: *RSA History*. <https://www.rsa.com/node.aspx?id=2760>
- [2] CryptoShop: *Passworte vs. OTP vs. PKI*. <http://www.cryptoshop.com/de/knowledgebase/authentication/passwortotppki/index.php>.
- [3] RSA: *SecurID<sup>®</sup> Token*. <https://www.rsa.com/node.aspx?id=1156>.
- [4] Kobil: *SecOVID Token*. <http://www.kobil.com/de/produkte/>.
- [5] VASCO: *DigiPass Token*. [http://vasco.com/products/digipass/digipass\\_index.aspx](http://vasco.com/products/digipass/digipass_index.aspx).
- [6] M'Raihi, Bellare, Hoornaert, Naccache, Ranen: *RFC4226: HOTP Algorithm*. <https://www.ietf.org/rfc/rfc4226.txt>
- [7] Krawczyk, Bellare, Canetti: *RFC2104: HMAC-SHA-1 Algorithm*. <https://www.ietf.org/rfc/rfc2104.txt>
- [8] M'Raihi, Machani, Pei, Rydell: *RFC6238: TOTP Algorithm*. <https://www.ietf.org/rfc/rfc6238.txt>
- [9] M'Raihi, Rydell, Bajaj, Machani, Naccache: *RFC6287: OCRA Algorithm*. <https://www.ietf.org/rfc/rfc6287.txt>
- [10] OATH: *Initiative for Open Authentication*. <http://www.openauthentication.org>
- [11] *mOTP: mobile One Time Passwords*. <http://motp.sourceforge.net>
- [12] *Google Authenticator*. <https://code.google.com/p/google-authenticator/>
- [13] LSE Leading Security Experts GmbH: *LinOTP 2*. <http://www.linotp.org>
- [14] Payment Card Industry - Data Security Standard: *PCI-DSS v2.0*. [https://www.pcisecuritystandards.org/security\\_standards](https://www.pcisecuritystandards.org/security_standards)
- [15] Josefson: *OATH Toolkit*. <http://www.nongnu.org/oath-toolkit/>
- [16] Lorez: login correct. Benutzerauthentifizierung mit OTP-Tokens. Tagungsband, Chemnitzer Linux-Tage 2011

# strace für bash-Versteher

Harald König

H.Koenig@science-computing.de  
<http://www.science-computing.de/>

Die UNIX-Shell ist ein sehr mächtiges und hilfreiches Werkzeug, welches leider nicht nur Linux-Anfänger immer wieder zur Verzweiflung treibt, wenn man ein paar einfache Grundregeln und Shell-Konzepte nicht kennt. Dieser Beitrag erklärt einige der wesentlichen Grundkonzepte in UNIX und der Shell, und illustriert und zeigt diese „Live“ mit Hilfe von `strace`. Dabei lernt man so nebenbei, wie nützlich `strace` beim Verstehen und Debuggen von Shell-Skripten und allen anderen UNIX-Prozessen sein kann.

## 1 Einleitung

Die UNIX/Linux-Shell (hier immer am Beispiel der `bash`) ist ein sehr einfaches, aber mächtiges Tool. Doch viel Benutzer haben oft Probleme, dass die Shell nicht ganz so will wie sie, Quoting, Wildcards, Pipes usw. sind immer wieder Ursache für Überraschungen und „Fehlverhalten“. Welches `echo` ist denn (wann?) das richtige?

```
$ echo Hallo           $ echo Hallo Chemnitz
$ echo 'Hallo'        $ echo Hallo Chemnitz
$ echo "Hallo"       $ echo 'Hallo Chemnitz'
```

Nach vielen Beobachtungen von solch schwierigen Entscheidungen in der freien Command-Line-Bahn war der letzte Auslöser für diesen Vortrag hier

```
# insmod *.ko
```

die Frage „Kann denn `/sbin/insmod` auch mit Wildcards umgehen?“.

## 2 UNIX-Grundlagen: `fork()` und `exec()`

Eines der universellen Prinzipien in UNIX ist der Mechanismus zum Starten aller neuer Programme: mit dem System-Call `fork()` wird ein neuer Prozess erzeugt, und dann das neue Programm mit dem System-Call `exec()` gestartet. Genau dies ist die Hauptaufgabe jeder UNIX-Shell, neben vielen einfacheren internen Shell-Kommandos. Dabei werden die Command-Line-Argumente für das neue Programm *einzel*n im `exec()`-Aufruf dem Programm übergeben:

```
main() { execl("/bin/echo", "programm", "Hallo", "Chemnitz 1", "", 0); }
main() { execl("/bin/echo", "programm", "Hallo", "Chemnitz", "1", 0); }
```

oder schöner

```

#include <unistd.h>
int main(int argc, char *argv[])
{
    char *const argv[] = { "programm", "Hallo", "Chemnitz 2", NULL };
    return execve("/bin/echo", argv , NULL );
}

```

Wie die Parameter für das neue Programm in diesen `exec()` System-Call kommen, ist Aufgabe der Shell. Wie das neue Programm dann diese einzelnen Argumenten (Optionen, Dateinamen, usw.) interpretiert und verarbeitet, liegt ausschließlich beim aufgerufenen Programm.

In Linux heißt der wirkliche Kernel-Call für `fork()` seit anno 1999 mit Kernel-Version 2.3.3 `clone()`, welcher von `fork()` in der `glibc` aufgerufen wird. Alle Varianten des System-Calls `exec()` rufen den einen wirklichen Kernel-Call `execve()` auf.

Daher ist es entscheidend zu verstehen, wie die Shell die eingegebene Command-Line verarbeitet und anschließend dem aufzurufenden Programm in `execve(...)` übergibt. Diese Übergabe im Kernel-Call lässt sich nun sehr schön mit `strace` überwachen und anzeigen, womit man exakt sehen kann, was denn die Shell aus der Eingabe wirklich gemacht hat und wie die Aufruf-Parameter des Programms wirklich aussehen.

### 3 Das Handwerkszeug zur Erkenntnis: `strace`

`strace` kennt viele Optionen und Varianten. Hier können nur die in meinen Augen wichtigsten angesprochen und gezeigt werden – die Man-Page (RTFM: `man strace`) birgt noch viele weitere Informationsquellen und Hilfen.

#### 3.1 Erste Schritte mit `strace`

Je nachdem, was man genau untersuchen will, kann man (genau wie mit Debuggern wie `gdb`) ein Kommando entweder mit `strace` neu starten, oder aber man kann sich an einen bereits laufenden Prozess anhängen und diesen analysieren:

```
$ strace emacs
```

bzw. für einen schon laufenden `emacs`

```
$ strace -p $( pgrep emacs )
```

und wenn es mehrere Prozesse gleichzeitig zu tracen gibt (z.B. alle Instanzen des Apache `httpd`):

```
$ strace $( pgrep httpd | sed 's/^-p/' )
```

Das an einen Prozess angehängte `strace -p ...` kann man jederzeit mit `CTRL-C` beenden, das untersuchte Programm läuft dann normal und ungebremst weiter. Wurde das zu testende Programm durch `strace` gestartet, dann wird durch `CTRL-C` nicht nur `strace` abgebrochen, sondern auch das gestartete Programm beendet.

strace gibt seinen gesamten Output auf stderr aus, man kann jedoch den Output auch in eine Datei schreiben bzw. die Ausgabe umleiten, dann jedoch inklusive dem stderr-Outputs des Prozesses (hier: emacs) selbst:

```
$ strace -o OUTFILE emacs
$ strace emacs 2> OUTFILE
```

Üblicherweise wird jeweils eine Zeile pro System-Call ausgegeben. Diese Zeile enthält den Namen der Kernel-Routine und deren Parameter, sowie den Rückgabewert des Calls. Der Output von strace ist sehr C-ähnlich, was nicht sehr wundern dürfte, da das Kernel-API von Linux/UNIX ja als ANSI-C-Schnittstelle definiert ist (meist didaktisch sinnvoll gekürzte Ausgabe):

```
$ strace cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 130 vars */]) = 0
open("/etc/HOSTNAME", O_RDONLY) = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28) = 28
read(3, "", 32768) = 0
close(3) = 0
close(1) = 0
close(2) = 0
exit_group(0) = ?
```

Selbst ohne C-Programmierkenntnisse lässt sich diese Ausgabe verstehen und vernünftig interpretieren. Details zu den einzelnen Calls kann man in den entsprechenden Man-Pages nachlesen, denn alle System-Calls sind dokumentiert in Man-Pages (man execve ; man 2 open ; man 2 read write usw.). Nur aus der Definition des jeweiligen Calls ergibt sich, ob die Werte der Argumente vom Prozess an den Kernel übergeben, oder aber vom Kernel an den Prozess zurückgegeben werden (bspw. den String-Wert als zweites Argument von read() und write() oben).

Für einige System-Calls (z.B. stat() und execve()) erzeugt strace mit der Option -v eine „verbose“ Ausgabe mit mehr Inhalt. Auf der Suche nach mehr Informationen (stat()-Details von Dateien, oder komplettes Environment beim execve()) kann dies oft weiterhelfen.

```
$ strace -e execve cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 130 vars */]) = 0
harald.science-computing.de

$ strace -v -e execve cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], ["LESSKEY=/etc/lesskey.bin", "MAN_PATH=/usr/local/man:/usr/loca...", "XDG_SESSION_ID=195", "TIME=\tt%E real,\tt%U user,\tt%S sy"... , "HOSTNAME=harald", "GNOME2_PATH=/usr/local:/opt/gnom"... , "XKEYSYMDB=/usr/X11R6/lib/X11/Xke"... , "NX_CLIENT=/usr/local/nx/3.5.0/bi"... , "TERM=xterm", "HOST=harald", "SHELL=/bin/bash", "PROFILERead=true", "HISTSIZE=5000", "SSH_CLIENT=10.10.8.66 47849 22", "VSCMB00T=/usr/local/scheme/.sche".. , "CVSROOT=/soft/.CVS", "GS_LIB=/usr/local/lib/ghostscrip"... , "TK_LIBRARY=/n ew/usr/lib/tk4.0", "MORE=-sl", "WINDOWID=14680077", "QTDIR=/usr/lib/qt3", "JRE [ ... many lines deleted ...] rap"... , "_=/usr/bin/strace", "OLDPWD=/usr/local/nx/3.5.0/lib/X"...]) = 0
```

## 3.2 Ausgabe von strace einschränken

Der Output von `strace` kann schnell *sehr* umfangreich werden, und das synchrone Schreiben der Ausgabe auf `stderr` oder auch eine Log-Datei kann erheblich Performance kosten. Wenn man genau weiß, welche System-Calls interessant sind, kann man die Ausgabe auf einen bzw. einige wenige Kernel-Calls beschränken (oder z.B. mit `-e file` alle Calls mit Filenames!), was sowohl den Programmablauf weniger verlangsamt als auch das spätere Auswerten des `strace`-Outputs erheblich vereinfacht. Allein die Option `-e` bietet sehr viel mehr Möglichkeiten. Hier nur ein paar einfache Beispiele, welche sehr oft ausreichen, alles Weitere dokumentiert die Man-Page:

```
$ strace -e open cat /etc/HOSTNAME
open("/etc/HOSTNAME", O_RDONLY) = 3

$ strace -e open,read,write cat /etc/HOSTNAME
open("/etc/HOSTNAME", O_RDONLY) = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28harald.science-computing.de
) = 28
read(3, "", 32768) = 0

$ strace -e open,read,write cat /etc/HOSTNAME > /dev/null
open("/etc/HOSTNAME", O_RDONLY) = 3
read(3, "harald.science-computing.de\n", 32768) = 28
write(1, "harald.science-computing.de\n", 28) = 28
read(3, "", 32768) = 0

$ strace -e file cat /etc/HOSTNAME
execve("/bin/cat", ["cat", "/etc/HOSTNAME"], [/* 130 vars */]) = 0
open("/etc/HOSTNAME", O_RDONLY) = 3
```

## 3.3 Mehrere Prozesse und Kind-Prozesse tracen

`strace` kann auch mehrere Prozesse gleichzeitig tracen (mehrere Optionen `-p PID`) bzw. auch alle Kind-Prozesse (Option `-f`) mit verfolgen. In diesen Fällen wird in der Ausgabe am Anfang jeder Zeile die PID des jeweiligen Prozesses ausgegeben. Alternativ kann man die Ausgabe auch ein jeweils eigene Dateien je Prozess bekommen (mit Option `-ff`).

Wenn man nicht genau weiß, was man denn eigentlich tracen muss, dann ist die Verwendung von `-f` oder `-ff` angesagt, da man ja nicht ahnen kann, ob evtl. mehrere (Unter-) Prozesse oder Skripte involviert sind. Ohne `-f` oder `-ff` könnten sonst beim Trace wichtige Informationen von weiteren Prozessen entgehen. In meinen einfachen Beispielen mit `emacs` ist dieser selbst auch schon ein Wrapper-Shell-Skript. Hier nun ein paar Varianten, die einen ein bissl grübeln lassen können, wie die `bash` intern so tickt:

```

$ strace -e execve bash -c true
execve("/bin/bash", ["bash", "-c", "true"], [/*...*/]) = 0

$ strace -e execve bash -c /bin/true
execve("/bin/bash", ["bash", "-c", "/bin/true"], [/*...*/]) = 0
execve("/bin/true", ["/bin/true"], [/*...*/]) = 0
$ strace -e execve bash -c "/bin/true ; /bin/false"
execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/*...*/]) = 0

$ strace -f -e execve bash -c "/bin/true ; /bin/false"
execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], [/*...*/]) = 0
execve("/bin/true", ["/bin/true"], [/*...*/]) = 0
execve("/bin/false", ["/bin/false"], [/*...*/]) = 0

$ strace -o OUT -f -e execve bash -c "/bin/true ; /bin/false"
$ grep execve OUT
1694 execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"], []) = 0
1695 execve("/bin/true", ["/bin/true"], [/*...*/]) = 0
1696 execve("/bin/false", ["/bin/false"], [/*...*/]) = 0

$ strace -o OUT -ff -e execve bash -c "/bin/true ; /bin/false"
$ grep execve OUT*
OUT.2155:execve("/bin/bash", ["bash", "-c", "/bin/true ; /bin/false"],[]) = 0
OUT.2156:execve("/bin/true", ["/bin/true"], [/*...*/]) = 0
OUT.2157:execve("/bin/false", ["/bin/false"], [/*...*/]) = 0

```

### 3.4 Lasst uns die bash tracen

Um nun einer interaktiven Shell „auf die Finger“ zu schauen, starten wir zwei Terminals. Im ersten Fenster fragen wir die PID der Shell, welche wir nun beobachten wollen, ab mit

```

$ echo $$
12345

```

und im zweiten Fenster starten wir dann mit dieser Information die Überwachung mit strace, welche alle aufgerufenen Programme ausgeben soll (ohne oder mit Option -v, je nach dem ob das Environment und lange/viele Argumente wichtig sind oder nicht):

```

$ strace -f -e execve -p 12345
$ strace -f -e execve -p 12345 -v

```

Im Folgenden wird in den Beispiel-Ausgaben immer zuerst die Shell-Eingabe und Ausgabe dargestellt, anschließend der zugehörige Output von strace.

und schon geht es los...

## 4 Globbing (Wildcards)

Zunächst sehen wir uns den Inhalt eines Verzeichnisses mit dem Kommando ls an. Einige Dateien werden hier mit einem Stern (\*) verziert. Grund hierfür ist die magisch aufgetauchte Option „-F“, welche von einem alias für ls stammt:

```

$ ls
echo1* echo1.c hello* hello.c hello.cpp hello.h test.c test.ko
execve("/bin/ls", ["ls", "-F"], [/*...*/]) = 0

```

```
$ type -a ls
ls is aliased to 'ls -F'
ls is /bin/ls
```

Um diesen alias zu umgehen, kann man entweder `/bin/ls` direkt aufrufen, oder `\ls` eingeben, beides hat die selbe Wirkung. Will man nur die C-Quell-Dateien sehen, so übergibt man `ls` das Argument `*.c`

```
$ /bin/ls
echo1 echo1.c hello hello.c hello.cpp hello.h test.c test.ko
execve("/bin/ls", ["ls"], [/*...*/]) = 0

$ \ls *.c
echo1.c hello.c test.c
execve("/bin/ls", ["ls", "echo1.c", "hello.c", "test.c"], [/*...*/]) = 0
```

doch mit `strace` sieht man, dass `ls` überhaupt nicht `*.c` als Parameter bekommt, sondern eine Liste aller einzelnen Dateinamen der C-Quellen. Das Expandieren von Wildcards (Globbing) wird bereits von der Shell übernommen, darum muss sich `ls` und alle anderen Programme nicht mehr selbst kümmern.

Wenn man das Expandieren der Dateinamen in der Shell mit Anführungszeichen (Quoting) verhindert, dann bekommt `ls` wirklich das Argument `*.c` übergeben und weiss damit nichts anzufangen:

```
$ \ls "*.c"
ls: cannot access *.c: No such file or directory
execve("/bin/ls", ["ls", "*.c"], [/*...*/]) = 0
```

Damit beantwortet sich auch die eingangs gestellte Frage nach `/sbin/insmod`:

```
# insmod *.ko
execve("/sbin/insmod", ["/sbin/insmod", "test.ko"], [/*...*/]) = 0

# insmod "*.ko"
insmod: can't read '*.ko': No such file or directory
execve("/sbin/insmod", ["/sbin/insmod", "*.ko"], [/*...*/]) = 0
```

`/sbin/insmod` kann mit Wildcards ebenso wenig umgehen wie `/bin/ls`, der Befehl `insmod` erwartet als Argument auf der Kommando-Zeile einen korrekten Dateinamen eines Kernel-Moduls. Bei `insmod` sollte man jedoch beachten, dass dieses Kommando nur ein einziges Kernel-Modul auf der Kommando-Zeile erwartet. Daher sollte man `insmod *.ko` so nur ausführen, wenn man sicher ist, dass sich nur genau ein Kernel-Modul `*.ko` im Verzeichnis befindet!

Das klassische Gegenbeispiel dieses ansonsten UNIX-üblichen Verhaltens ist der Befehl `find` mit seiner Option `-name`. Hier kann man nach `-name` ein „Pattern“ mit Wildcards angeben, nach welchem gesucht wird. Es führt zu Fehlern oder überraschendem Verhalten, wenn die Wildcards nicht gequotationet und deshalb von der Shell expandiert werden. Im ersten Beispiel wird ausschliesslich nach `test.ko` gesucht und nicht nach allen Kernel-Modulen (Überraschung), im zweiten Beispiel gibt es eine kryptische Fehlermeldung von `find`, da man dessen Syntax nicht eingehalten hat:

```
$ find .. -name *.ko
../dir/test.ko
execve("/usr/bin/find", ["find", "..", "-name", "test.ko"], [/*...*/]) = 0
```

```
$ find .. -name *.c
find: paths must precede expression: hello.c
Usage: find [-H] [-L] [-P] [-Olevel] [-D help|tree|search|stat|rates|opt|exec] [path...] [expression]
execve("/usr/bin/find", ["find", "..", "-name", "echo1.c", "hello.c", "test.c"], [/*...*/]) = 0
```

Hier muss man mit Anführungszeichen oder Backslash das Expandieren in der Shell verhindern, damit find für -name das richtige Pattern übergeben bekommt:

```
$ find .. -name \*.ko
execve("/usr/bin/find", ["find", "..", "-name", "*.ko"], [/*...*/]) = 0
../dir/test.ko
../dir2/test2.ko

$ find .. -name "*.c"
../dir/echo1.c
../dir/hello.c
../dir/test.c
../dir2/test2.c
execve("/usr/bin/find", ["find", "..", "-name", "h*.c"], [/*...*/]) = 0
```

## 5 Quoting

Schon die Beispiele zu Wildcards zeigen, dass man in der Shell ab und an „quoten“ muss, fragt sich nur wann, und wie man es richtig macht. In diesem Zusammenhang muss man natürlich die Zeichen mit Sonderfunktionen in der Shell kennen, die man evtl. im Einzelfall umgehen will.

Hierzu gehören, wie schon gesehen, die Wildcards für Pattern (\* und ?), das Dollarzeichen (\$) für Variablenamen u.ä., Größer-, Kleiner- und Pipe-Zeichen (< > |) für Ein- und Ausgabe-Umleitung sowie Pipes, runde Klammern für Sub-Shells, das &-Zeichen für Hintergrund-Prozesse, das Ausrufezeichen (!) für die Shell-History, die sogenannten White-Space-Characters (Space, Tab, Zeilenende), der Backquote (‘) für die Ausgaben-Einfügung, sowie natürlich auch die Quoting-Zeichen (\ ' ") selbst, und vermutlich noch ein paar weitere hier vergessene Zeichen.

Die Shell kennt drei Methoden, Zeichen mit Sonderfunktionen in der Shell, wie z.B. die Wildcards, zu „quoten“: den Backslash (\), sowie die einfachen (‘) und doppelten (") Anführungszeichen (engl. *quotes*, daher der Begriff Quoting). Die Funktion der drei Quoting-Zeichen ist eigentlich auch schnell erklärt:

Der Backslash (\) wirkt auf das nächste Zeichen und hebt dessen Sonderfunktion auf.

In Zeichenketten innerhalb von einfachen Anführungszeichen (‘) gibt es keinerlei Sonderfunktionen mehr, mit der einzigen Ausnahme des (‘) selbst, welches die gequotete Zeichenkette beendet. Auch der Backslash hat keine Sonderfunktion mehr. Daher kann auch das Single-Quote in solchen Zeichenketten *nicht* gequotet werden. Explizite Single-Quotes kann man nur außerhalb gequoteter Strings bzw. innerhalb doppelter Anführungszeichen verwenden.

In Zeichenketten innerhalb von doppelten Anführungszeichen (") sind noch einige wenige Sonderzeichen aktiv: \$ für Variablenamen, Backquote (‘) für die Ausgaben-Einfügung und das Ausrufeszeichen bei der interaktiven Eingabe, sowie der Back-

slash um diese Zeichen und auch sich selbst quoten zu können. Leerzeichen, Tabular-Zeichen und Zeilenenden bleiben in beiden Arten der Anführungszeichen erhalten.

echo ist in der bash ein *interner* Befehl und wird dadurch nicht von strace erfasst, auch nicht mit einem Backslash davor, wie bei den Alias von ls:

```
$ echo Hallo Chemnitz
Hallo Chemnitz

$ \echo Hallo Chemnitz
Hallo Chemnitz
```

Um Leerzeichen zu erhalten muss gequotet werden, ob mit einfachen oder doppelten Anführungszeichen oder mit Backslashes ist hier unerheblich:

```
$ /bin/echo Hallo Chemnitz
Hallo Chemnitz
execve("/bin/echo", ["/bin/echo", "Hallo", "Chemnitz"], [/*...*/]) = 0

$ /bin/echo "Hallo Chemnitz"
Hallo Chemnitz
execve("/bin/echo", ["/bin/echo", "Hallo Chemnitz"], [/*...*/]) = 0

$ /bin/echo 'Hallo Chemnitz'
Hallo Chemnitz
execve("/bin/echo", ["/bin/echo", "Hallo Chemnitz"], [/*...*/]) = 0

$ /bin/echo Hallo\ \ \ Chemnitz
Hallo Chemnitz
execve("/bin/echo", ["/bin/echo", "Hallo Chemnitz"], [/*...*/]) = 0
```

Doch auch schon richtig gequotete Leerzeichen können schnell wieder verschwinden, wenn man einmal das Quoten vergisst

```
$ a="A B C"
$ /bin/echo $a
A B C
execve("/bin/echo", ["/bin/echo", "A", "B", "C"], [/*...*/]) = 0

$ /bin/echo "$a"
A B C
execve("/bin/echo", ["/bin/echo", "A B C"], [/*...*/]) = 0

$ /bin/echo '$a'
$a
execve("/bin/echo", ["/bin/echo", "$a"], [/*...*/]) = 0
```

und auch Wildcards müssen stets gebändigt und im Quote-Zaum gehalten werden, solange man deren Expansion nicht wünscht:

```
$ b="mit find kann man nach *.c suchen..."
$ /bin/echo "$b"
mit find kann man nach *.c suchen...
execve("/bin/echo", ["/bin/echo", "mit find kann man nach *.c suchen.."], []) = 0

$ /bin/echo $b
mit find kann man nach echo1.c hello.c test.c suchen...
execve("/bin/echo", ["/bin/echo", "mit", "find", "kann", "man", "nach",
"echo1.c", "hello.c", "test.c", "suchen.."], []) = 0
```

Benötigt man auf einer Kommandozeile sowohl die Wirkung von einfachen als auch doppelten Anführungszeichen, so muss man diese entsprechend abwechselnd öffnen

und schliessen, oder man verwendet Backslashes wo möglich:

```
$ echo 'Die Variable "a" wird mit $a ausgelesen und hat den Wert \''$a''
Die Variable "a" wird mit $a ausgelesen und hat den Wert 'A B C'
execve("/bin/echo", ["/bin/echo", "Die Variable \"a\" wird mit $a ausgelesen
und hat den Wert 'A B C'"], [/*...*/]) = 0

$ echo "Die Variable \"a\" wird mit \$a ausgelesen und hat den Wert \''$a\"
Die Variable "a" wird mit $a ausgelesen und hat den Wert 'A B C'
execve("/bin/echo", ["/bin/echo", "Die Variable \"a\" wird mit $a ausgelesen
und hat den Wert 'A B C'"], [/*...*/]) = 0

$ /bin/echo Die Variable \"a\" wird mit \$a ausgelesen und hat den Wert \''$a\"
Die Variable "a" wird mit $a ausgelesen und hat den Wert 'A B C'
execve("/bin/echo", ["/bin/echo", "Die", "Variable", "\"a\"", "wird", "mit",
"$a", "ausgelesen", "und", "hat", "den", "Wert", "'A B C'"], []) = 0
```

und noch einmal kurz die schon besprochenen Wildcards

```
$ /bin/echo *.c
echo1.c hello.c test.c
execve("/bin/echo", ["/bin/echo", "echo1.c", "hello.c", "test.c"], [/*...*/]) = 0

$ /bin/echo *.c
*.c
execve("/bin/echo", ["/bin/echo", "*.c"], [/*...*/]) = 0

$ /bin/echo '*.c'
*.c
execve("/bin/echo", ["/bin/echo", "*.c"], [/*...*/]) = 0
```

## 6 Welche Login-Skripte wurden ausgeführt

... ist mit strace auch ganz allgemein zu prüfen und zu verifizieren. Entweder kann man der entsprechenden Shell vertrauen, dass diese sich z.B. mit deren Option --login wirklich identisch zu einem „echten“ Login verhält. Mit dieser Annahme reicht es, eine File-Liste erstellen und diese zu analysieren. Im gezeigten Beispiel bleibt FILELIST unsortiert, denn auch die Reihenfolge der Login-Skripte ist natürlich interessant:

```
$ strace -o OUT -e open -f bash --login -i
exit
$ grep -v ENOENT OUT | grep \" | cut -d\" -f2 > FILELIST1
$ strace -o OUT -e file -f bash --login -i
exit
$ grep \" OUT | cut -d\" -f2 > FILELIST2
```

```
$ egrep "^/etc|${HOME}" FILELIST1
/etc/profile
/etc/bash.bashrc
/etc/venus/env.sh
/etc/bash.local
/home/koenig/.bashrc
/etc/bash.bashrc
/etc/venus/env.sh
/etc/bash.local
/home/koenig/.bash_profile
/home/koenig/.profile.local
/home/koenig/.bash_history
/etc/inputrc
/home/koenig/.bash_history
```

Wenn man der Shell nicht so ganz traut, dann sollte man die „Quelle“ eines Logins selbst tracen! Als Nicht-root kann man dies z.B. noch mit `xterm -ls` versuchen. Oder als root direkt den SSH-Daemon an Port 22 oder ein `getty`-Prozess mit allen Folge-Prozessen tracen:

```
$ lsof -i :22
COMMAND PID USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
sshd     2732 root   3u  IPv4 15551    0t0  TCP *:ssh (LISTEN)
$ strace -p 2732 -ffv -e file -o sshd.strace ...
```

## 7 Conclusion

Was man mit `strace` noch alles über die Skript-Ausführung erfahren und das gesamte UNIX-/Linux-System lernen kann, ist Stoff für weitere Vorträge. Der Phantasie bietet sich hier viel Raum für weitere Einsatzmöglichkeiten.

Nun einfach viel Spaß mit der Shell, und Erfolg und neue UNIX-Erkenntnisse beim stracen!

## Literatur

- [1] RTFM: `man strace bash`
- [2] <http://linux.die.net/man/1/bash>                      Man-Page von bash
- [3] <http://linux.die.net/man/1/strace>                    Man-Page von strace

# Tic-Tac-Toe Reloaded — Ein Mikrocontroller-Projekt mit Funkübertragung

Axel Wachtler, Jörg Wunsch, Matthias Vorwerk

{axel.joerg.matthias}@uracoli.de

<http://uracoli.nongnu.org/ct2012>

Das altbekannte Tic-Tac-Toe-Spiel wird in einem ATmega128RFA1-Mikrocontrollerprojekt zu neuem Leben erweckt. Es werden kapazitive und resistive Touch-Sensoren realisiert sowie eine drahtlose Datenübertragung in das Spiel integriert. Hierbei standen ein kostengünstiger und einfacher Aufbau der Hardware sowie eine leicht nachvollziehbare Software-Realisierung im Vordergrund.

## 1 Einführung

Mikrocontroller sind Bestandteil verschiedenster Geräte des täglichen Lebens. Auf dem Markt existiert eine riesige Auswahl dieser interessanten Schaltkreise, angefangen von kleinsten 4-Bit-Varianten (z. B. in Fernbedienungen) über 8-Bit-Controller unterschiedlicher Größe und Komplexität (z. B. für Ladegeräte, Kühlschränke, Waschmaschinen) bis hin zu leistungsfähigen 32-Bit-Architekturen, die virtuelle Adressräume und Speicherverwaltung zur Verfügung stellen, sodass auf ihnen Multitasking-Betriebssysteme laufen können. Mittlerweile sind Mikrocontroller mit integrierter Funkchnittstelle erhältlich. Damit können auch über längere Distanzen Informationen drahtlos übertragen werden. Ein Beispiel dafür ist der Controller ATmega128RFA1 von Atmel.

Das Ziel des Projektes ist es, eine Hardware zu entwickeln, die sowohl ein grundsätzliches Verständnis der Beschaltung dieses interessanten Bauteils als auch den Aufbau und die Funktionsweise der zum Betrieb nötigen Software vermittelt. Ansprechende Interaktionsmöglichkeiten für den Nutzer waren ebenso ausschlaggebend wie niedrige Hardwarekosten, die durch ein minimalistisches Schaltungsdesign erreicht wurden.

Als Anwendungsbeispiel wird das jahrhundertalte Spiel *Tic-Tac-Toe* („Drei gewinnt“) [2] implementiert. Dabei spielen zwei Spieler auf einem 3x3-Spielfeld und belegen abwechselnd ein noch freies Feld mit einem eigenen Symbol (z. B. Kreuz, Kreis). Ziel ist es, als erster eine komplette Zeile, Spalte oder Diagonale mit dem eigenen Symbol zu belegen. In der Mikrocontroller-Realisierung bedient jeder der beiden Spieler ein 3x3-Eingabefeld auf seiner eigenen Platine. Als visuelle Rückmeldung werden jeweils 3x3-zweifarbige LED verwendet, die unter dem Eingabefeld aufleuchten. Derjenige Spieler, welcher als erster eine Zeile, Spalte oder Hauptdiagonale mit „seiner“ LED-Farbe belegt hat, gewinnt. Die Spielereingaben werden über eine Funkverbindung zwischen beiden Platinen übertragen.

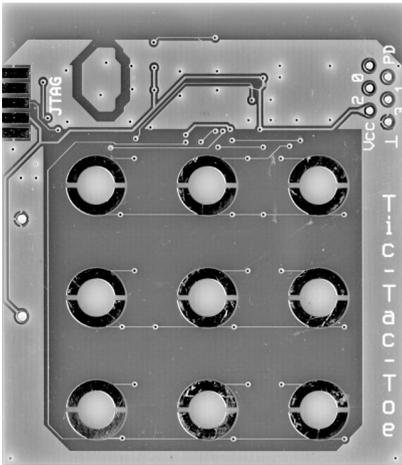


Bild 1: Leiterplattenoberseite mit resistiven Sensorflächen

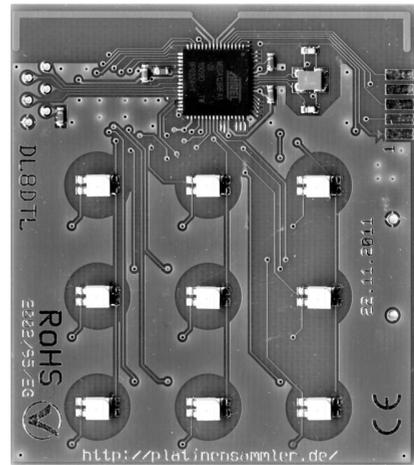


Bild 2: Leiterplattenunterseite mit Mikrocontroller und LED

## 2 Das Herzstück: Der ATmega128RFA1

Als Mikrocontroller wird ein 8-Bit-AVR verwendet, da dieser Typ weit verbreitet ist und auch eine hervorragende Unterstützung durch Open-Source-Entwicklungstools erfährt (`avr-gcc`, `avr-libc`, `avr-gdb`, `avrdude`). Der für dieses Projekt ausgewählte ATmega128RFA1 [3] ist ein 8-Bit-RISC-Controller mit 128 KiB Flash-Speicher, 4 KiB EEPROM sowie 16 KiB SRAM. Er besitzt zusätzlich eine Funkschnittstelle nach dem *IEEE-802.15.4*-Standard für das 2,4-GHz-ISM-Band, mit der Daten mit 250 kBit/s (Sondermodi bis zu 2 MBit/s) drahtlos übertragen werden können. Weiterhin enthält er einen 6 Bit breiten sowie vier 8 Bit breite Ein-/Ausgabeports, welche für die Ansteuerung der Leuchtdioden sowie der kapazitiven und resistiven Sensorpads eingesetzt werden. Weitere Einzelheiten zum Schaltkreis sind im Datenblatt [3] nachzulesen.

## 3 Verwendete Hardware

Die Hardware besteht aus zwei eigens für diesen Zweck entwickelten, nahezu gleichen Platinen. Jede Platine enthält:

- einen AVR Mikrocontroller ATmega128RFA1 mit *IEEE-802.15.4*-Funkschnittstelle von Atmel,
- ein 3x3-Touchfeld aus Leiterzügen, wobei eine Platine aus resistiven und die zweite Platine aus kapazitiven Touch-Sensoren besteht,

- 9 Zweifarb-LED, die ebenfalls als 3x3-Matrix auf der Unterseite der Platine angeordnet sind,
- einen Batteriehalter,
- einen 16-MHz-Quarz,
- einige SMD-Kondensatoren,
- verzinnte Anschlusspads für das JTAG-Programmierinterface,
- Anschlusspads zu unbenutzten, digitalen Ein-/Ausgabepins (UART und I<sup>2</sup>C-Interface) für eigene Erweiterungen.

Die gewählte Minimalbeschaltung zeigt zum einen, dass ein derartiger Mikrocontroller, abhängig von den konkreten Anforderungen, mit sehr wenig Außenbeschaltung auskommen kann und erleichtert zum anderen das Verständnis der Schaltung.

Durch die Verwendung von kapazitiven bzw. resistiven Sensoren konnten einerseits Drucktaster eingespart werden, andererseits bietet dies die spannende Herausforderung, Algorithmen zur zuverlässigen Erkennung von Fingerdrücken zu finden, was zusätzlichen Softwareaufwand erfordert. Dieses Verlagern von Hardwareaufwand in die Software ist ein durchaus üblicher und oft gewählter Ansatz.

Bild 1 und Bild 2 zeigen Platinenansichten. Die Sensorflächen befinden sich auf der Platinenoberseite. Die LED auf der Bauteilseite durchscheiden die Leiterplatte und hinterleuchten so die Sensoren.

#### 4 Die Schaltung

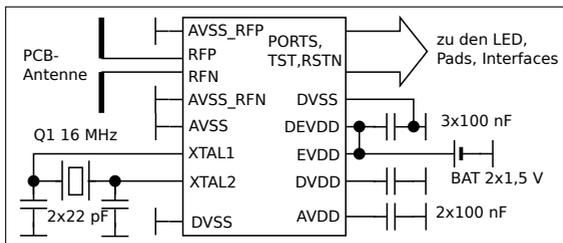


Bild 3: Außenbeschaltung des ATmega128RFA1 (ohne LED/Touchmatrix)

Bild 3 zeigt die Beschaltung des Mikrocontrollers. Neben einem 16-MHz-Quarz und zwei 22-pF-Lastkondensatoren, die mit der internen Oszillatorschaltung die notwendige stabile Frequenzreferenz für die Funkkommunikation liefern, sind lediglich noch fünf Stützkondensatoren für die Versorgungsspannung sowie neun Leuchtdioden zur Anzeige des Spielstandes als externe Bauelemente nötig. Der Mikrocontroller selbst arbeitet mit einem 1-MHz-Takt, der durch einen internen RC-Oszillator mit nachgeschaltetem Teiler bereitgestellt wird.

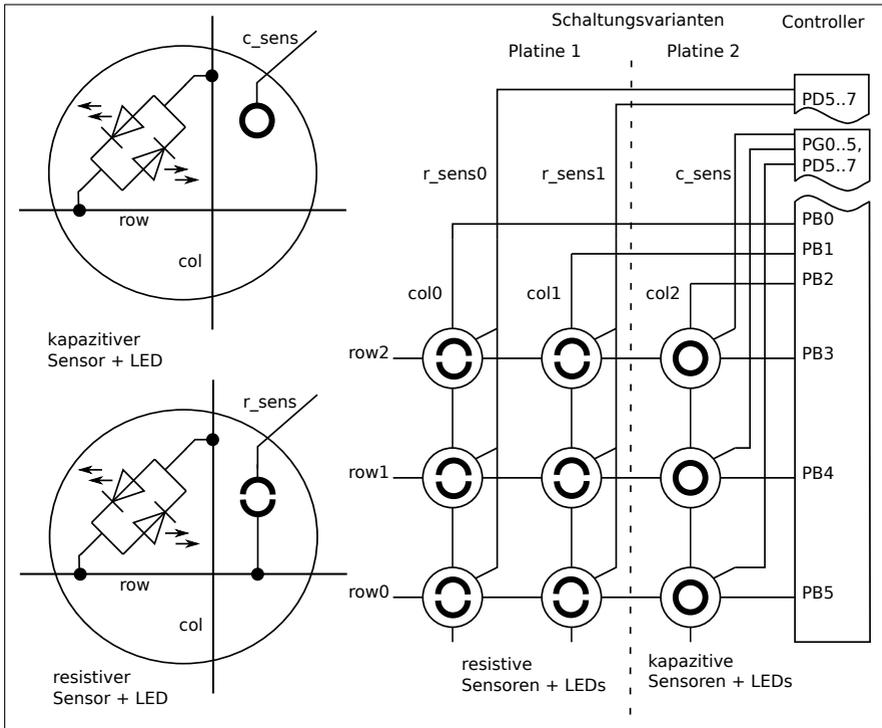


Bild 4: Beschaltung der Ein-/Ausgabematrix für beide Platinenvarianten

Die Antenne wird als PCB-Antenne in Form von Leiterzügen auf der Platine realisiert. Auf das normalerweise in der Grundbeschaltung vorgesehene Filter zur Unterdrückung unerwünschter Oberwellen an den HF-Pins wurde verzichtet. Diese Vereinfachung erfüllt die Forderungen der Norm ETSI EN 300 440-1 (unerwünschte Ausstrahlungen oberhalb 1 GHz maximal  $1 \mu\text{W}$ ) gerade noch ausreichend; zur Erhöhung des Sicherheitsabstandes ist ein Betrieb mit leicht reduzierter Sendeleistung ( $-5 \text{ dBm}$ ) ratsam. Ein Betrieb außerhalb des Geltungsbereiches der europäisch harmonisierten Regelungen ist jedoch mit dieser einfachen Anordnung in der Regel nicht zulässig.

Die Schaltung der LED- und Sensormatrix für beide Platinenvarianten ist in Bild 4 dargestellt. Die Pins 3, 4, 5 von Port B bilden die drei Zeilenleitungen, die Pins 0, 1, 2 die Spaltenleitungen der  $3 \times 3$ -LED-Matrix. Die Leuchtdiodenmodule befinden sich an den Kreuzungspunkten der Zeilen- und Spaltenleitungen. Durch High/Low-Ausgaben an den entsprechenden Bits dieses Ports kann jeweils eine der beiden antiparallel geschalteten unterschiedlich farbigen Leuchtdioden eingeschaltet werden. Aufgrund des geringen maximalen Stroms der Ausgangstreiber im Schaltkreis ist es möglich, auf die ansonsten notwendigen Vorwiderstände zu verzichten.

Für die Touch-Sensoren wurden zwei unterschiedliche Ansätze gewählt, um die Workshop-Teilnehmer mit verschiedenen Realisierungsvarianten für ein gegebenes Problem bekannt zu machen.

Die kapazitiven Sensoren wurden als Leiterzüge in Form von breitrandigen Kreisen realisiert, um einerseits eine möglichst große Kapazitätsfläche zu erreichen und andererseits ein Durchscheitern der auf der Gegenseite angeordneten LED zu gewährleisten. Alle neun Sensorflächen werden separat an die Pins 5, 6, 7 (Port D) und 0 – 5 (Port G) geführt.

Im Gegensatz zu den kapazitiven Sensoren werden für den resistiven Ansatz zwei Leitungen pro Sensor benötigt. Um das Platinendesign einfach zu halten und die Zahl der notwendigen Ein-/Ausgabeports zu beschränken, werden die 3x3-Sensoren ebenfalls matrixförmig durch Zeilen- und Spaltenleitungen angesteuert, wobei die Sensoren an den Kreuzungspunkten angeordnet sind. Als Zeilenleitungen werden die für die LED-Ansteuerung verwendeten Leitungen mit genutzt (Pins 3, 4, 5 von Port B); die Spaltenleitungen werden mit den Pins 5, 6, 7 von Port D verbunden.

Die kompletten Schaltungsunterlagen sind auf der Projektwebseite zu finden.

### Die kapazitiven Touch-Sensoren

Die Wirkungsweise der kapazitiven Sensoren wird in Bild 5 gezeigt. Vor Beginn eines Abfragevorgangs ( $t < t_0$ ) ist der Schalter  $S_d$  geschlossen. Die Kapazität  $C_{in} = C_{pin} + C_{pad}$ , die sämtliche gegen Masse wirkenden Kapazitäten zusammenfasst, ist entladen, und am Pin  $c\_sens$  liegt eine Spannung von 0 Volt an.

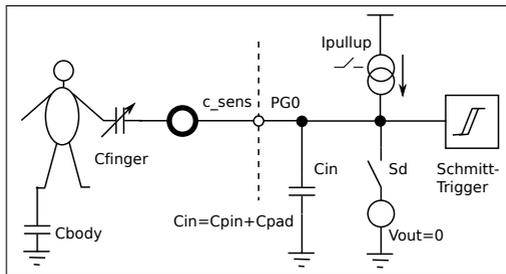


Bild 5: Wirkungsprinzip kapazitiver Sensor

Zu Beginn der Messung bei  $t = t_0$  wird  $S_d$  geöffnet (Abschalten des Ausgangstreibers) und ein Pullup-Strom  $I_{pullup}$  beginnt, die Kapazität aufzuladen. Wegen  $I = Cdu/dt$  ist der messbare Spannungsanstieg je Zeiteinheit am Pin durch die Kapazität bestimmt. Die Zeit  $t' - t_0$  bis zum Schalten des Schmitt-Triggers ist somit ein Maß für die Gesamtkapazität am Pin.

Bei Berührung des Sensors wird eine weitere Kapazität  $\frac{1}{(1/C_{finger} + 1/C_{body})}$  parallel

zu  $C_{in}$  wirksam, was in einer längeren Aufladezeitdauer  $t'' - t_0$  resultiert. Für die Erkennung einer Berührung wird in der Software eine Zeit  $t_{sample}$  mit  $t' < t_{sample} < t''$  als Entscheidungskriterium genutzt.

Entscheidend für die sichere Erkennung einer Berührung des Sensors ist  $t'' \gg t'$  und somit eine möglichst große Kapazitätsänderung.

$C_{body}$  ist wegen  $C \sim A/d$  aufgrund der Körperoberfläche hinreichend groß. Eine hohe Kapazität  $C_{finger}$  wird erreicht, indem der direkt auf die Kapazitätsflächen aufgebraute Lötstopplack mit einer Dicke von wenigen  $10 \mu m$  ein sehr dünnes Dielektrikum zwischen Finger und Sensorfläche bildet. Durch diese Anordnung kann die gewünschte hohe Kapazitätsänderung erreicht werden.

Es ist zu beachten, dass  $I_{pullup}$  stark betriebsspannungsabhängig ist ( $5 \dots 30 \mu A$ ). Wegen  $du/dt = I_{pullup}/C_{total}$  muss die relative Kapazitätsänderung im Berührungsfall deutlich größer als die zu erwartende relative Variation des Pullup-Stroms sein, um dennoch eine sichere Berührungserkennung gewährleisten zu können. Gegebenenfalls sind Kalibriermessungen denkbar, die eine Anpassung von  $t_{sample}$  in Abhängigkeit der gemessenen Versorgungsspannung  $V_{cc}$  und der Zeit  $t'$  zur Folge haben. Für die derzeitige Software-Realisierung ist dies nicht vorgesehen.

### Die resistiven Touch-Sensoren

Das Funktionsprinzip der resistiven Sensoren ist in Bild 6 dargestellt.

Die beiden Sensorkontakte werden durch den symbolischen Widerstand  $R_{finger}$  nachgebildet. Dessen Widerstandswert kann, wenn beide Kontakte durch einen Finger berührt werden, abhängig von mehreren Faktoren (Hautfeuchte, Druckstärke, ...) stark variieren und liegt zum Teil im zweistelligen Megaohm-Bereich. In der Kapazität  $C_{in} = C_{pin} + C_{pad}$  werden alle parasitären Eingangskapazitäten der Leitung  $r_{sens}$  zusammengefasst. Eine weitere parasitäre Kapazität  $C_{par}$  parallel zu den beiden Sensorkontakten bildet mit  $C_{in}$  einen kapazitiven Spannungsteiler. Da  $C_{par} \ll C_{in}$  gilt, wird im Einschaltmoment  $t = t_0$  die Schaltschwelle des Triggers nicht erreicht. Im folgenden wird  $C_{par}$  nicht weiter betrachtet.

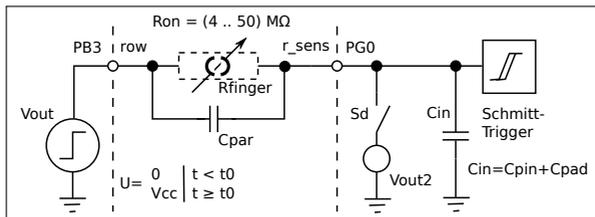


Bild 6: Wirkungsprinzip resistiver Sensor

Vor Beginn der Messung bei  $t < t_0$  beträgt die Ausgangsspannung des treibenden Ausgangspins (Leitung row) 0 Volt. Die Leitung  $r_{sens}$  ist ebenfalls als Ausgang und

auf 0 Volt geschaltet ( $S_d$  geschlossen).  $C_{par}$  und  $C_{pin} + C_{pad}$  sind entladen.

Zu Beginn der Messung ( $t = t_0$ ) wird  $S_d$  geöffnet und das Sense-Pin (PG0 in Bild 6) wird als Eingang betrieben. Gleichzeitig wird die Zeilenleitung row auf  $V_{cc}$  geschaltet (Ausgangspegel High). In Abhängigkeit von  $R_{finger}$  wird nun  $C_{in}$  auf die Spannung  $U(t - t_0) = V_{cc}(1 - e^{-(t-t_0)/\tau})$  mit  $\tau = R_{finger}C_{in}$  aufgeladen. Bei Erreichen der Schaltschwelle des Schmitt-Triggers nach der Zeit  $t' - t_0$  schaltet das entsprechende Eingangsbit auf 1.

Für angenommene Werte  $R_{finger} = 50 \text{ M}\Omega$  und  $C_{in} \approx 20 \text{ pF}$  wird die Schaltschwelle  $V_{cc}/2$  nach  $t' - t_0 = \tau \ln 2 = 0,69 \text{ ms}$  erreicht. Für  $R_{finger} = 1 \text{ M}\Omega$  (minimaler Überbrückungswiderstand durch den Finger) ergibt sich eine Zeit von  $t' - t_0 = 13,86 \text{ us}$ . Eine sichere Erkennung erfordert mithin u. U. eine relativ lange Messzeit.

## 5 Software-Realisierung

Die Software wird in der Programmiersprache C geschrieben. Ein wichtiger Bestandteil sind die Routinen zur Abfrage der Touch-Sensoren. Die zeitlich exakte Messung der kurzen Aufladezeit der kapazitiven Sensoren wird durch 16-faches sequenzielles Einlesen der Input-Port-Information realisiert und erfordert die Verwendung von Inline-Assembler-Code.

Da die Leuchtdioden matrixförmig verschaltet sind, ist eine zeilenweise, zeitmultiplexe Ansteuerung notwendig. Diese und die zyklische Abfrage der Berührungssensoren werden in einer Interruptroutine ausgeführt, die per Timer gestartet wird.

Die Datenübertragung zwischen den beiden Knoten erfolgt über die eingebaute Funkchnittstelle des ATmega128RFA1, um beiden Spielern den aktuellen Spielstand synchron anzeigen zu können. Die Ansteuerung der integrierten Sende- und Empfangsbaugruppen erfordert prinzipiell einen nicht unerheblichen Softwareaufwand, der den Rahmen dieses Projektes bei Weitem sprengen würde.

Mit dem  **$\mu$ racoli-Projekt** [1] existiert für diesen und verwandte Schaltkreise von Atmel eine freie Softwarebibliothek, die die Low-Level-Ansteuerung der Funkbaugruppen dieser Schaltkreise übernimmt und High-Level-Funktionen `radio_send_frame()` und `usr_radio_receive_frame()` zur Übertragung kompletter Datensätze zur Verfügung stellt.

Die vollständige Software zu diesem Projekt steht unter einer modifizierten BSD-Lizenz und ist auf der zugehörigen Webseite [1] zu finden.

## 6 Übersetzen und Inbetriebnahme

Zum Entwickeln, Übersetzen und Programmieren der Controller wurde die AVR-Toolchain für Linux verwendet, bestehend aus dem GCC-Compiler `avr-gcc`, dem Linker aus den `GNU binutils` und der `avr-libc`. Der Programmieradapter wird mit dem Programm `avrdude` angesteuert. Beispiele für integrierte Entwicklungsumgebungen,

die ebenso genutzt werden können (und dann teilweise oder vollständig auf der genannten Toolchain aufsetzen), sind das *AVR Studio* [4] (nur für Windows), *Eclipse* oder *Code::Blocks*.

Zur Programmierung der Mikrocontroller können die *AVR-Dragon*-Boards von Atmel zum Einsatz kommen. Deren mitgeliefertes Kabel wird mit einem Zwischenadapter verbunden, welcher direkt auf die vorgesehenen verzinnten JTAG-Anschlussfahnen der Platinen gesteckt wird.

Neben der Programmierung ist es auch möglich, über das JTAG-Interface *On Chip Debugging* durchzuführen. Hierzu stellt das Tool *avriace* eine Verbindung zwischen dem Schaltkreis und dem *avr-gdb* (Debugger) her, sodass schrittweise Programmabarbeitung und die Abfrage interner Register möglich ist.

## 7 Ausblick

Die entwickelten Platinen ermöglichen es, erste Hardware- und Programmiererfahrungen mit AVR-Mikrocontrollern zu sammeln. Da die Platinen neben neun zweifarbigen LED und neun Eingabefeldern auch die Möglichkeit der Funkkommunikation nach dem *IEEE-802.15.4*-Standard bieten, sind eine Vielzahl anderer Einsatzmöglichkeiten denkbar. Die realisierte Version des Tic-Tac-Toe-Spiels soll lediglich als Anregung dienen und Lust auf eigene Experimente machen. Über Anregungen, Fragen und aktive Hilfe freut sich das *uracoli*-Team.

## Literatur

- [1] *uracoli - The Micro Controller Radio Communication Library*. Homepage  
<http://uracoli.nongnu.org>
- [2] *Tic Tac Toe*. Wikipedia  
[http://de.wikipedia.org/wiki/Tic\\_Tac\\_Toe](http://de.wikipedia.org/wiki/Tic_Tac_Toe)
- [3] *ATmega128RFA1* Herstellerinformationen  
[http://www.atmel.com/dyn/products/product\\_card.asp?part\\_id=4692](http://www.atmel.com/dyn/products/product_card.asp?part_id=4692)
- [4] *Atmel AVR Studio*. Produktseite  
<http://www.atmel.com/avrstudio>
- [5] A. Wachtler, J. Wunsch, M. Vorwerk, *Drahtlose Sensordatenerfassung und -verarbeitung mit Linux*. Tagungsband Chemnitzer Linuxtage 2011. Chemnitz 2011. 13-20

# Ubiquitous Computing [ju:'bɪk.wɪ.təs kəm'pju:tɪŋ]

## - Chancen und Risiken -

**Klaus Knopper**

uc@knopper.net

<http://knopper.net/clt2012/>

Der Begriff „Ubiquitous Computing“ kennzeichnet die Allgegenwärtigkeit von Informationstechnologie im täglichen Leben. Neu hierbei ist, dass die eigentliche Technik mehr oder weniger „unsichtbar“ für die Nutzer in die Umgebung integriert ist und die klassischen „Computer“ (selbst die noch immer populären Mini-Notebooks) nach und nach völlig zu verschwinden scheinen.

Datenerfassung und Automatisierung von Abläufen mit integrierter Sensorik, drahtloser Informationsübertragung und Auswertung in vernetzten Computersystemen, die nicht einmal mehr direkt von Menschen „bedient“ werden müssen, wecken einerseits Hoffnungen bezüglich der Erleichterung von Alltagsaufgaben und erweiterte Kommunikationsmöglichkeiten für viele Menschen, andererseits sind damit auch Gefahren durch die intransparente Erfassung, Speicherung und Vernetzung von Daten verbunden (Stichwort RFID, automatisierte biometrische Erkennung von Personen, Erfassung von Fahrzeugen etc.).

Der Vortrag gibt eine Übersicht über den aktuellen Stand der Technik beim Ubiquitous Computing und zeigt die bedeutende Rolle offener Software- und Hardwarearchitekturen in der Entwicklung auf. Ein besonderer Fokus liegt dabei auf der Sicherheit, Nachhaltigkeit und Zuverlässigkeit von Systemen, die beim Ubiquitous Computing zum Einsatz kommen und es werden Szenarien aufgezeigt, die jeden unvermeidlichen Teilnehmer am Ubiquitous Computing zumindest nachdenklich stimmen könnten.

## 1 Ubiquitous Computing<sup>1</sup>

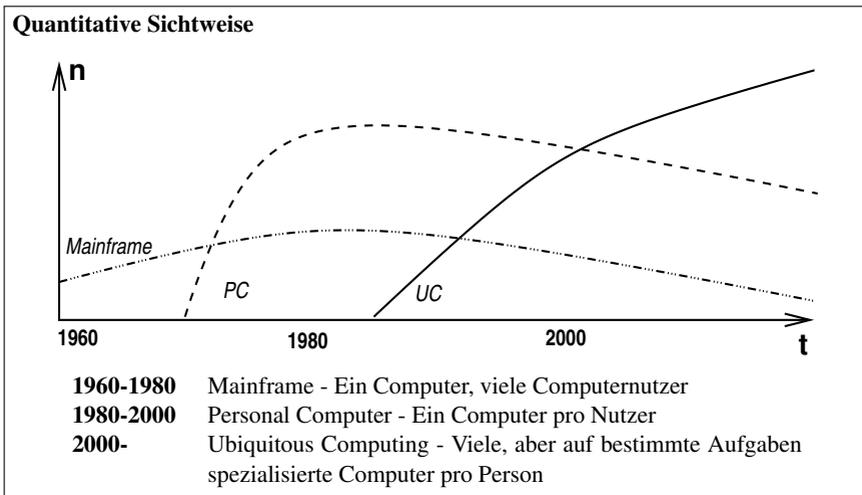
Der Begriff *Ubiquitous Computing* wurde erstmals 1988 von *Mark Weiser* [2] verwendet und 1991 in seinem Paper *The Computer for the 21st Century (1991)* vertieft, die Idee der „unsichtbaren, vernetzten Computer“ reicht jedoch weiter zurück [3].

Daten werden mit (ggf. unsichtbar) in die Umgebung integrierter Sensorik erfasst, Information wird drahtlos in vernetzte Computersysteme übertragen und ausgewertet und Arbeitsabläufe mit Hilfe dieser Daten gesteuert und automatisiert. Dabei ist eine Interaktion durch Computernutzer oder Servicepersonal meist weder erforderlich, noch erwünscht.

Erklärtes Ziel des Ubiquitous Computing ist hierbei, in allen Lebensbereichen sowie in Industrie und Handel als Unterstützung zu dienen und Menschen von komplizierten Aufgaben zu entlasten.<sup>2</sup>

<sup>1</sup> „Die Allgegenwärtigkeit (Ubiquität, engl. ubiquity) der rechnergestützten Informationsverarbeitung“ [1]

<sup>2</sup> Weiser selbst sah 1991 PDAs (Personal Digital Assistant, ☞ heute „Smartphones“) als eine Vorstufe des UC



## Verwandtschaft

Neben „Ubiquitous Computing“ werden in der Fachliteratur weitere Begriffe, teils alternativ und mit großen inhaltlichen Überschneidungen verwendet, jedoch mit unterschiedlichem Schwerpunkt.

<b>Internet der Dinge</b>	Computergestützte Identifikation und Verwaltung von realen Objekten (z.B. RFID [14])
<b>Pervasive Computing</b>	„allesdurchdringende Vernetzung des Alltags“, kommerzielle Anwendung der automatisierten Datenerfassung und -auswertung (E-Commerce unterstützt durch u.a. RFID, Data Mining)
<b>Ambient Intelligence<sup>3</sup></b>	„Intelligente“ Technik soll v.a. den Menschen unterstützen, um dessen Leistungsfähigkeit und Lebensqualität zu erhöhen.
<b>Organic Computing</b>	„selbstorganisierende“ verteilte Rechnerstrukturen
<b>Wearable Computing</b>	Elektronische „Gadgets“ in Kleidung oder am Körper (Identifikation & Authentifizierung, Hilfsmittel, Entertainment)

Zwei weitere Begriffe werden oft im Zusammenhang mit UC verwendet: „*Smart Dust*“ (miniaturisierte Computer mit Sensoren, die zur Erkundung und Überwachung eingesetzt werden), und „*Nomadic Computing*“, was eher die „Notebook- und Smartphone-Generation“ kennzeichnet, die immer und überall persönlich Internet-Dienste nutzt. Sowohl auf der Seite

<sup>3</sup>„Intelligenz“ hat je nach Namensraum (allg. Sprachgebrauch, IT, Militär) eine sehr unterschiedliche Bedeutung.

der Anwendung (Smartphones/Tablets mit Android oder einem vollständigem Desktop Anwendungssystem) als auch auf der Seite der hierfür notwendigen, verteilten Netzwerkinfrastruktur spielen GNU/Linux und andere Open Source Systeme nicht nur im embedded Bereich (Accesspoints, Router, Gateways) eine Hauptrolle aufgrund ihrer Anpassungsfähigkeit, Performance und Zuverlässigkeit auf unterschiedlichster Hardware. Viele Accesspoints lassen sich beispielsweise mit neuer Linux-basierter Firmware ausstatten, die dann Netzwerkfestplatten, Streaming von Multimedia-Content und IP-Telefonie, Drucker- und Filesharingdienste oder Teilnahme an der Cloud ermöglicht.

*Next Generation Media* ist ein weiterer mit Ubiquitous Computing verwandter, allerdings ausschließlich in Deutschland verwendeter Begriff und kennzeichnet ein Technologieprogramm des Bundesministeriums für Wirtschaft und Technologie mit Fokus auf Konsumelektronik, Logistiknetze, Produktionsanlagen und Gesundheitsversorgung. [8]

### **Drahtlose Verbindung**

Eine Kernkomponente beim Ubiquitous Computing ist die feste oder lose Kopplung der Agenten über verschiedene Kommunikationswege. Die historisch gesehen früheste Variante ist die kabelgebundene Direktverbindung, die heute noch bei bandbreitenintensiven Anwendungen bzw. als „Backbone“ dient, während vor allem bei mobilen netzwerkfähigen Geräten WLAN und Bluetooth zur Vernetzung eingesetzt werden. Während letztere vor allem für den kontrollierten, paketorientierten Informationsfluss in Mikroprozessorgesteuerten Geräten eingesetzt werden, spielen aus Gründen der Effizienz und Kostenreduktion einfacher aufgebaute Technologien in der Nahfeldkommunikation (NFC) eine zunehmend größere Rolle. Hierbei ist vor allem die Radio Frequency Identification und ihre Anwendung zu nennen.



Bildquelle: [14]

RFID wird für die drahtlose Übertragung von Daten zwischen elektronischen „Etiketten“ oder Smartcards und einem Computersystem verwendet. Neben Kennzeichnungen können auch bidirektionale Kommunikationsverfahren verwendet werden, in der *Anwendungen* auf der Smartcard vom Transponder gesendete Daten verarbeiten und ein Ergebnis zurückschicken, was häufig zu Verschlüsselungszwecken oder Manipulation von Datenströmen (Dekoder) verwendet wird. Ob hierbei nur Schlüssel oder Fragmente übertragen werden, oder die komplette Datenverarbeitung über die Smartcard läuft, hängt von der verfügbaren Rechenleistung ab. Oft verfügen RFID-Chips nur über rein logische Schaltungen, aber es sind mittlerweile auch Smartcards mit eigenem Betriebssystem und komplexere Anwendungen im Einsatz, die die gleiche Technologie nutzen. Wenn Anwendungen auf Smartcards auch nicht den Funktionsumfang üblicher Anwenderprogramms besitzen, so gibt es doch immerhin Open Source Werkzeuge zur Unterstützung der Entwicklung und Tests, z.B. als Plugin für Eclipse [4].

Je nach Einsatzzweck kommen passive oder aktive Transponder, mit oder ohne Verschlüsselung, mit oder ohne (Wieder-)Beschreibbarkeit zum Einsatz. Seit 2000 werden passive 13,56 MHz RFID-Transponder in Form flexibler Etiketten zur Mediensicherung in Bibliotheken verwendet.

Der breiteste Einsatz findet sich derzeit in der Logistik, um Warensendungen zu verfolgen und Lagerbestände zu verwalten. RFID-Chips sind außerdem in allen seit dem 1. November 2005 ausgestellten deutschen Reisepässen sowie ab dem 1. November 2010 in allen Personalausweisen enthalten.

## **2 Einsatzgebiete**

UC hat in den letzten 10 Jahren zu einem Technologiewechsel in vielen Einsatzgebieten geführt, in denen nun neben einer sehr weit gehenden Automatisierung auch eine gewisse Eigendynamik der Entwicklungsrichtung wahrzunehmen ist, die auch gesellschaftliche Fragen aufwirft.

### **Personenidentifikation**

In Reisepass oder ID-Karte mit RFID wird für das Auslesen eine Applikation mit Basic Access Control (BAC) verwendet, die als Teil der Authentifizierung die optisch eingescannte maschinenlesbare Zone (MRZ) verwendet, dadurch ist ein Zugriff auf die gespeicherten personenbezogenen Daten zumindest nicht ohne weiteres „im Vorübergehen“ möglich. Über eine Anwendung auf dem RFID-Chip ist es auch möglich, sich mittels Challenge-Response bei verschiedenen Diensten online anzumelden, ein geeignetes Lesegerät vorausgesetzt.

Chip-Implantate bei Menschen sind derzeit eine eher seltene Variante der Personenidentifizierung, bei Haus- und Nutztieren wird dieses Verfahren jedoch bereits angewendet.

Anstelle oder zusätzlich zur Identifizierung über mitgeführte *RFID-Tags* werden bei Personen, was ebenfalls als Ubiquitous Computing angesehen werden kann, in der Biometrie eindeutige Merkmale wie Fingerabdrücke, Konturen des Ohrs oder Retina-Signaturen verwendet und sensorisch erfasst. Über festgelegte Ähnlichkeitskriterien wird dann der Bezug zur Identität hergestellt.

### **Handel und Verleih**

Aufklebbare Etiketten mit RFID-Chip werden häufig zur Warensicherung eingesetzt, können aber auch zur Inventarisierung oder automatischen Buchung verwendet werden. [14]

### **Transport und Logistik**

Die automatische oder halbautomatische Erfassung und Buchung von Waren mittels RFID oder optisch erfasstem Barcode (auch Matrix-Barcode, z.B. QR-Code [5]) können die Effizienz in Warenwirtschaftssystemen und im Supply Chain Management steigern.

Die an allen Autobahnen in Deutschland vorhandenen Kontrollbrücken für die Mautgebühr von LKWs erfassen Fahrzeugnummern und Fahrzeugprofile (3D-Kontur) *aller* durchfahrenden Fahrzeuge. Bei LKWs mit On-Board-Unit werden zusätzlich Kenndaten per Infrarotsignal übermittelt.



Bildquelle: [http://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Mautbr%C3%BCcke01\\_2009-04-13.jpg/800px-Mautbr%C3%BCcke01\\_2009-04-13.jpg](http://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Mautbr%C3%BCcke01_2009-04-13.jpg/800px-Mautbr%C3%BCcke01_2009-04-13.jpg)

## **Im Unternehmen**

In Unternehmen und Betrieben wird UC zu Identifikationszwecken von Mitarbeitern (Zugangskontrolle, Facilitymanagement) im Supply-Chain-Management, v.a. der Ortsverfolgung und Protokollierung von Waren (Lagerverwaltung, Kühlkette, Qualitätsmanagement, Teilverfolgung, Wiederbeschaffung) sowie im Customer-Relationship-Management (Bestellungsverfolgung, Bezahlung, Kundenidentifikation, Kundenbetreuung, Vertriebskontrolle, Wartung) eingesetzt. [7]

Hierbei spielen auch plattformunabhängige Notifikationssysteme, teils auf Basis von LAMP mit browserbasierten mobilen Clients eine Rolle, die z.B. einen möglichen dringenden Handlungsbedarf dem zuständigen Projektverantwortlichen mitteilen.

## **Hausautomatisierung**

Durch die Vernetzung von Geräten durch Steueranlagen können Hausbewohner unterstützt werden (Licht ein-/ausschalten, Fensterrollos öffnen und schließen, Vorräte überwachen).

Beispiel: In einigen Hotels werden entnommene Waren aus der Minibar automatisch erkannt und nachbestellt.

Auch die Überwachung und Steuerung bei Abwesenheit ist möglich (vergessene Herdplatte ausschalten, Strom-/Wasserverbrauch überwachen, Videorecorder oder Staubsaugerroboter per Internet oder zeitgesteuert aktivieren).



Bildquelle:

[http://upload.wikimedia.org/wikipedia/de/thumb/6/6f/Reinigungsroboter\\_tcm\\_100.JPG/800px-Reinigungsroboter\\_tcm\\_100.JPG](http://upload.wikimedia.org/wikipedia/de/thumb/6/6f/Reinigungsroboter_tcm_100.JPG/800px-Reinigungsroboter_tcm_100.JPG)

Hier ist auch Z-Wave als technische Umsetzung einer geräteübergreifenden Automatisierung per Funk zu nennen (<http://de.wikipedia.org/wiki/Z-Wave>).

### **Gesundheitswesen und Pflege**

Über drahtlos vernetzte, am Körper getragene Sensortechnik können pflegebedürftige oder hilflose Personen von einem Dienstleister betreut werden, und erhalten so mehr Selbstständigkeit und Sicherheit in ihrem eigenen Zuhause.

Durch „intelligente“ Sensortechnik kann ein Sturz oder außergewöhnliche Verhaltensweisen automatisch erkannt und an eine Zentrale gemeldet werden, so dass kein manueller Notruf ausgelöst werden muss.

Blinde Menschen können sich durch Geräte mit Sprachausgabe und Braille-Displays orientieren. Für motorisch eingeschränkte Personen ist Sprach- und Gestenerkennung eine Möglichkeit der direkten Mensch-Maschine-Kommunikation.

## **3 Ubiquitous Computing und GNU/Linux**

Ubiquitous Computing benötigt zur Verwaltung und Auswertung der anfallenden Daten ein für die teils sehr spezialisierten Aufgaben ein geeignetes Betriebssystem, daher ist hier GNU/Linux schon weiter verbreitet als auf dem klassischen Desktop-Universalcomputer.

### **Buchungs- und Warenwirtschaftssysteme, Back Office**

Geschäftsvorgänge bedürfen der Speicherung, Auswertung und Präsentation durch Softwaresysteme, die auch von technisch unbedarftem Personal bedient werden können, wofür sich

auf das jeweilige Einsatzgebiet optimierte Systeme z.B. auf LAMP-Basis hervorragend eignen. Oft werden hierbei auf einem zentralen Server in PHP, Perl oder Python geschriebene Anwendungen oder Java-Applets mit Zugriff auf die gesammelten Daten mit einer browserbasierten Oberfläche eingesetzt, zunehmend wird für die Oberflächenprogrammierung auch AJAX eingesetzt. Für den Benutzer, der das System bedient, spielt Art der auf Server und Clients eingesetzten Software, ob Open Source oder proprietär, zwar eher eine untergeordnete Rolle; für Systemarchitekten und Programmierer ist jedoch eine Präferenz zugunsten Freier Software durch die umfangreichen verfügbaren Frameworks und durch die weitgehend plattformunabhängige Anpassbarkeit gegeben.

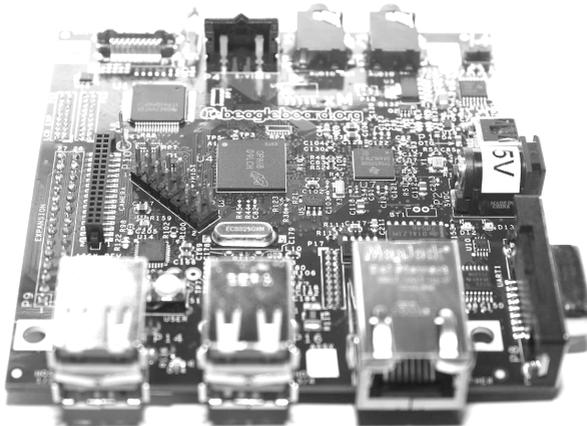
Beispiel: Auf den Kassen- und Abrechnungssystemen einiger Discounter sind immer öfter für aufmerksame Kunden „Pinguine“ sichtbar.<sup>4</sup>

### Embedded Linux

Wenige Hersteller spezialisierter Geräte sprechen gerne über „interne Angelegenheiten“ wie die auf ihren Geräten laufende Firmware, daher ist es schwierig, eine genaue Statistik mit dem Anteil von Open Source anzugeben.

Eine Ausnahme stellen Smartphones dar, bei denen die Systemsoftware und die damit verbundene Verfügbarkeit von Anwendungen ein publiziertes Verkaufsmerkmal ist, hier erfährt das Linux-basierte *Android* derzeit die messbar höchste Verbreitung (Marktanteil 53% im 3. Quartal 2011 lt. Gartner-Analyse vom November 2011).

In einem Großteil der komplexeren im Ubiquitous Computing auftauchenden „intelligenten Geräte“ kommen Mini-Computerboards zum Einsatz, auf denen z.B. ein Linux-Kernel mit einem sehr spartanischen System (oft *busybox*) und auf die Hardware angepassten Modulen die Steuerungsaufgaben übernimmt. ☞ Embedded GNU/Linux



Bildquelle: [http://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/BeagleBoard\\_xM.JPG/1280px-BeagleBoard\\_xM.JPG](http://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/BeagleBoard_xM.JPG/1280px-BeagleBoard_xM.JPG)

<sup>4</sup>Was nicht alleine zum Dekorationszweck dient, sondern oft durch das „Branding“ des Softwareherstellers und IT-Dienstleisters vorgegeben ist.

Der Einsatz von Open Source Software als Basis für ihre Anwendungen erspart den Herstellern einerseits Aufwand für die Neuentwicklung der Plattform und stellt eine solide und flexibel anpassbare Entwicklungsbasis dar, andererseits bleiben lizenzrechtliche Fragen genauso wenig aus wie bei proprietärer Software:

- ◇ Bedeutet das Mitliefern der Software auf dem Gerät eine „Verbreitung der Software“ im Sinne der GNU GENERAL PUBLIC LICENSE, oder handelt es sich um eine hardwaregebundene „Firmware“, die gar nicht von einem Anwender direkt „genutzt“ wird? Muss man dementsprechend jedem Käufer Zugang zum Quelltext gewähren, und der Konkurrenz damit auch selbst entwickelte Algorithmen offenlegen? <sup>138</sup> Für einige Geräte, bei denen der Anwender sehr wohl direkt mit der Software arbeiten kann (z.B. Accesspoints) gibt es bereits einschlägige Streitfälle und gerichtliche Auseinandersetzungen, die die Offenlegung der auf GPL-Code basierenden Quelltexte fordern und auch Bearbeitungen durch Dritte erlauben. [9]
- ◇ Wie sind proprietäre Module und Programmteile von Zulieferern, die nicht unter eine Open Source Lizenz gestellt werden können, zu trennen von Open Source Bestandteilen in der Systemsoftware? <sup>139</sup> Lösbar durch Anbieten des Source Code nur für die Freien Bestandteile, allerdings dürfen die proprietären Teile nicht so untrennbar mit dem Open Source System verbunden sein, dass ohne sie gar nichts mehr funktioniert.
- ◇ Lassen sich gemischt proprietäre/Open Source Systeme in kritischen Bereichen einsetzen, bei denen es um hohe Betriebssicherheit, z.B. in medizinischen Bereich geht, wenn eine vollständige Analyse auf Quelltextbasis durch die proprietären Teile nicht möglich ist?
- ◇ Wie können zertifizierte Systeme aktualisiert werden (z.B. dringende Fehlerbehebungen, neuer Kernel), ohne dass jedesmal neu zertifiziert werden muss?

## 4 Risiken

*„Komplexe Informationstechnologie versagt in dem Moment, in dem wir beginnen, uns darauf zu verlassen.“ [11]*

Ubiquitous Computing ist heute in vielen Bereichen eine unverzichtbare Komponente. Studien zur Technikfolgen-Abschätzung [6] versuchen, die Risiken in aktuellen (an der jeweils aktuellen Technologie orientierten) Einsatzszenarien aufzuzeigen und abzuwägen.

### Zuverlässigkeit der Sensorik

„Intelligente Gegenstände“ im informationstechnischen Sinne verfügen nicht über Intelligenz, teilweise nicht einmal über Mechanismen zur Fehlererkennung in „außergewöhnlichen“ Situationen. Als generelle Regel gilt: Je komplexer ein System ist, desto anfälliger ist es für Fehlinterpretation der erfassten Daten, ein „Plausibilitäts-Check“ findet in realen Implementationen selten statt. So werden beispielsweise bei Geschwindigkeitskontrollen mitunter durch

Softwarefehler (nicht selten auch Bedienfehler) Kinderwagen, Nutztiere oder Häuser mit über 90 km/h „geblitzt“. Allerdings sind in diesem Fall nicht die vermessenen Objekte, sondern die Meßeinrichtungen „fehlerhaft intelligent“, da sie durch unzureichend robuste Algorithmen gesichert sind, die eigentlich als Filter für Daten außerhalb der Spezifikation gedacht sind (oder durch das Fehlen ebensolcher Filter).

### **„Der gläserne Bürger“**

Obwohl in der für RFID & Co. verwendeten Near Field Communication nur wenige Zentimeter Abstand zwischen den Kommunikationspartnern liegen, ist es mit geeigneter Sende- und Empfangstechnik möglich, diesen „Minimalabstand“ auch auf mehrere Meter zu erweitern, so dass zumindest diejenigen Daten, die ohne vorherige Autorisation verfügbar sind, auch „im Vorübergehen“ ausgelesen werden können. Dies betrifft zwar nach aktuellem Stand der Technik nicht die personenbezogenen Daten, die im maschinenlesbaren Ausweis gespeichert sind, aber z.B. solche, die für eine einfache Warenidentifizierung verwendet werden.

Bei unsicher konfigurierten Handys oder anderen persönlichen Elektronischen Geräten ist es möglich, per WLAN oder Bluetooth sämtliche Daten auszulesen oder zu verändern. Unabhängig davon ist immer eine, wenn auch nicht sehr zuverlässige, Ortung über die Funkzelle oder eine IP-Adresse möglich.

Über die Verknüpfung gesammelter Daten ist die Erstellung eines Persönlichkeitsprofils möglich, das nicht nur zu Marketingzwecken, sondern schlimmstenfalls auch zum Identitätsdiebstahl verwendet werden kann. Die betroffene Person hat keine Kontrolle über die über sie gesammelten Daten.

☞ *Verlust der informationellen Selbstbestimmung*

### **Verlässlichkeit erfasster Daten**

Biometrische Merkmale wie Fingerabdrücke sind einfach zu fälschen [13], aber auch bei den Originalen steigt die Fehlerrate z.B. bei alterungsbedingten Veränderungen rapide an. Unverschlüsselte RFID-Kommunikation zur Authentifizierungszwecken lässt sich durch Replay-Attacken fälschen. Fehlerkorrekturmechanismen wie Checksummen sind nicht eindeutig: Zu einem geringen Prozentsatz kann es zu „zufälligen“ Treffern bzw. falsch erkannten Identifikationen kommen. Bei Hardwarefehlern in Sensor oder Transponder werden keine brauchbaren Daten geliefert und es muss auf Alternativinformation zugegriffen werden.

Das Vertrauen in die Verlässlichkeit elektronischer Information ist, da subjektiv, oft ungerechtfertigt hoch, so dass mögliche Fehlerquellen nicht berücksichtigt werden. Dadurch bleiben mitunter auch „offensichtliche“ Fehler unentdeckt oder werden als „systeminhärent“ akzeptiert - und ignoriert.

### **„99% Langeweile und 1% panische Angst“<sup>5</sup>**

Durch die Automatisierung von Arbeitsabläufen ergibt sich in den meisten Fällen eine starke Eingrenzung von Einsatzgebieten für Menschen auf lediglich das, was nicht (oder noch nicht)

---

<sup>5</sup>Feedback von Piloten und Operateuren, die mit neuen Bedien- und Automatisierungskonzepten konfrontiert wurden.  
[12]

automatisierbar ist. Wenn der Arbeitseinsatz für den Menschen jedoch nur auf das Eingreifen in Not- und Ausnahmefällen reduziert wird, bekommt die eigentlich als Unterstützung gedachte Technik aus psychologischer Sichtweise plötzlich ein „feindliches Gesicht“: ☞ Scheinbarer oder tatsächlicher Kontrollverlust durch nicht beeinflussbare Automatismen, und „Nicht wissen, was im Ausnahmefall zu tun ist.“

### **Folgen der Vernetzung disjunkter Information**

Durch die Vernetzung von eigentlich unabhängiger Information aus unterschiedlichen Quellen entsteht Potenzial für Fehlinterpretation aufgrund einer angenommenen, aber nicht tatsächlich vorhandenen Kausalität.

Beispiele:

- ◇ Der PKW einer Person wird von einer Verkehrskamera nahe Basel registriert. Der Führerschein des PKW-Besitzers wird zur gleichen Zeit in einer Verkehrskontrolle in Zweibrücken gescannt. Zur gleichen Zeit wird mit der Kreditkarte des PKW-Besitzers eine Restaurantrechnung in Berlin bezahlt.  
☞ Wurde das Auto gestohlen, der Führerschein, die Kreditkarte?
- ◇ Die Alarmanlage eines Elektronik-Geschäftes schlägt beim Betreten oder Verlassen des Geschäftes an, weil ein RFID-Etikett in einem getragenen Schuh, der in einem anderen Geschäft erworben und bezahlt wurde, eine bestimmte Kennung sendet. ☞ Kein Abgleich zwischen verschiedenen Einsatzszenarien der RFID-Technik, „Überflutung“ mit automatisch erfasster Information

☞ Da der Computer bei einer automatischen Auswertung einem Ereignis keine *Relevanz in der realen Welt* zuordnen kann, können auch - vollautomatisch - Fehlentscheidungen unterschiedlichen Ausmaßes getroffen werden.

Weiterhin werden Systeme durch starr festgelegte Algorithmen unflexibel. Ein eher harmloses Beispiel: In einem Fastfood-Restaurant ist es nicht möglich, um 9:30 Uhr morgens einen „Cheeseburger“ zu bestellen, da das Kassensystem und die Steuerungssoftware zu diesem Zeitpunkt automatisch auf die „Frühstückskarte“ eingestellt ist, die nur eine eingeschränkte Produktwahl zulässt.

### **Datenschutz und Datensicherheit**

Die beim UC anfallenden Daten sind oft personenbezogener Natur, und bedürfen besonderem Schutz vor unberechtigtem Zugriff, der bei der Übertragung und Speicherung nicht immer gewährleistet ist. Das Einholen einer Einverständniserklärung der betroffenen Personen ist problematisch, da für diese meist wegen der „Unsichtbarkeit“ der Sensoren gar nicht ersichtlich ist, dass und welche Daten erfasst werden.

Die gesammelten Daten, z.B. durch die zur Maut-Kontrolle aufgestellten Kontrollbrücken, könnten prinzipiell verwendet werden, um Bewegungsprofile von Fahrzeugen und Personen zu erstellen. Derzeit dürfen die Daten nur stichprobenweise für die Kontrolle der bezahlten

LKW-Maut verwendet werden, und die Kontrollbrücken sind nur zu bestimmten Tageszeiten in Betrieb.

### **UC und die Cloud (Technik und Sicherheit)**

Oft ist der Wirkungsbereich im UC lokal begrenzt, und eine dauerhafte Speicherung der Daten findet nicht statt. Wenn die beim Ubiquitous Computing anfallenden Daten jedoch zentral ausgewertet und gespeichert werden sollen, liegt es nahe, die notwendige Rechenleistung und Datenspeicher aus einer ebenfalls vernetzten Struktur zu beziehen (v.a. private clouds [10]). Um den sicheren Transfer und die Weiterverarbeitbarkeit der Information zu gewährleisten, werden standardisierte Protokolle verwendet (XML, http(s), ISO/IEC 15961/15962, SSL).

Die Technologie der sicheren Datenübertragung und Datenhaltung mit kryptographischen Verfahren ist heute in der Theorie ausgereift und mathematisch verifizierbar. Die relevanten Algorithmen sind als Open Source in den verschiedensten Programmiersprachen mit Bibliotheken und Entwicklungsumgebungen als Open Source verfügbar.

Bei Public Key Verfahren, die heute auch für die Verschlüsselung von Webseiten beim Online-Banking u.ä. eingesetzt werden, wird ein Schlüsselpaar verwendet, mit dem nur der jeweils komplementäre Schlüssel in der Lage ist, die Information aus den zuvor mit dem Primärschlüssel gesicherten Daten wieder zugänglich zu machen. Da die Schlüssel selbst bei der Kommunikation nicht übertragen werden, ist das „Knacken“ der verschlüsselten Übertragung mit handelsüblichen Computersystemen nur schwer möglich.

Trotzdem gelingt es Angreifern in manchen Fällen, unautorisierten Zugriff auf Daten zu erhalten, und diese auch zu manipulieren (s. nächster Abschnitt), was zu Industriespionage, Zerstörung von IT-Infrastruktur und Identitätsdiebstahl führen kann.

### **Sicherheits-Schwachstellen bei der verschlüsselten Datenübertragung und -speicherung im Ubiquitous Computing**

Durch *Phishing* [15] können Zugangsschlüssel in die Hände unberechtigter Personen gelangen, dies wird oft verursacht durch menschliches Versagen bei der Beurteilung der Vertrauenswürdigkeit von Webseiten oder Gesprächspartnern bei der elektronischen Kommunikation. Auf bereits manipulierten Client-Systemen können die Daten vor der Verschlüsselung abgefangen werden, wenn nicht sogar bei implementationsbedingender fehlender Verschlüsselung im Klartext mitgelesen werden (unzureichende Validierung der beteiligten Kommunikationskomponenten). Bei Verschlüsselungsalgorithmen mit Public Key Verfahren ist der Dienstanbieter (Cloud-Betreiber) i.d.R. im Besitz des Private Key oder eines „Gruppenschlüssels“, mit dem ein Teil der bei ihm gelagerten Daten genau wie mit dem Primärschlüssel dechiffriert werden kann, was in einigen Ländern sogar gesetzlich geregelte Pflicht ist. Hierdurch können einerseits die Strafverfolgungsbehörden „verdächtige“ Inhalte oder Protokolldaten mit Hilfe des Anbieters untersuchen, andererseits fördert die unvorsichtige Handhabung (z.B. zentrale Speicherung von Private Key „Backups“ auf Rechnern in öffentlich zugänglichen Netzen) spektakuläre „Datenschutzpannen“. Dem Anbieter eines Cloud-Dienstes wird oft eine zu hohe Vertrauensstellung eingeräumt. Anwender verifizieren in Unkenntnis der Sicherheitsmechanismen oft den von einem Cloud-Dienst präsentierten Public Key nicht, oder ignorieren Warnungen, die besagen, dass der Key nicht von der gleichen Autorität signiert wurde wie

der Private Key. Hierdurch ist es einem „Man in the Middle“ möglich, Daten des Anwenders abzufangen, bevor sie zum eigentlichen Dienst gelangen. Auf der Gegenseite werden Daten oft zwar verschlüsselt übertragen, aber dann doch unverschlüsselt gespeichert.

### **Wie kann der Einsatz von Open Source dabei helfen, Risiken zu reduzieren?**

Offene Systeme und offene Standards erleichtern die System-Analyse und die Wahrung von Transparenz, und erleichtern es, die im UC meist nicht sichtbaren beteiligten IT-Systeme und deren Funktionsweise zu verstehen.

Für den Nutzer (bei Desktop-Computern spricht man vom „Endanwender“) der im UC verwendeten Technologie spielt es zwar immer weniger eine Rolle, dass das heimische, vollständig drahtlos vernetzte und Internet-angebundene Multimedia-System inklusive Fernseher unter GNU/Linux läuft, weil eher die einfache Bedienbarkeit bzw. die Benutzeroberfläche im Vordergrund steht (die bei Freier Software natürlich gegeben ist [16]), jedoch gilt auch im Bereich der stark spezialisierten Computersysteme, dass Open Source als Basis viel an Flexibilität und Erweiterungsmöglichkeiten bietet und einer künstlichen Einschränkung von Nutzungsmöglichkeiten digitaler Inhalte (s.a. „Digital Rights Management“) entgegenwirkt. Systematische Fehler in einer Implementation sind durch die öffentliche Verfügbarkeit der Quelltexte leichter zu entdecken und zu beheben, das Vertrauen in die Funktionsweise und die Nutzerakzeptanz wird bei einem verifizierbaren offenen System deutlich gestärkt.

## **5 Zusammenfassung**

- ◇ UC: Datenerfassung und Automatisierung von Abläufen mit (ggf. unsichtbar) in die Umgebung integrierter Sensorik, drahtloser Informationsübertragung und Auswertung in vernetzten Computersystemen.
- ◇ UC verdrängt zunehmend den klassischen „Desktop“-Universalcomputer.
- ◇ UC kann in fast allen Lebensbereichen Assistenz bieten und Abläufe vereinfachen und beschleunigen.
- ◇ Datenschutz und Datensicherheit sind technisch realisierbar, jedoch aufgrund von Interessenskonflikten oft eingeschränkt, oder fehlerhaft implementiert.
- ◇ Relationsbildung gesammelter und vernetzter Daten erlaubt die Erstellung von Persönlichkeitsprofilen und Protokollierung der Aufenthaltsorte von Personen.
- ◇ Die automatische Aus- und Bewertung von Daten kann zu Fehlinterpretationen und Fehlentscheidungen führen, die nur schwer korrigierbar sind (Kontrollverlust).
- ◇ UC: Bei der technischen Implementation spielen Linux und Open Source eine große Rolle, wenn es um komplexe Aufgaben geht, die sich nicht mit reinen Logikbausteinen realisieren lassen.
- ◇ GNU/Linux und Open Source bieten über offene Schnittstellen und öffentliche Verfügbarkeit des Quelltextes Nachhaltigkeit und Transparenz, und fördern unter günstigen Voraussetzungen Vertrauen in die Zuverlässigkeit und Erweiterbarkeit der vernetzten Systeme.

## Quellenangaben und Literaturverzeichnis

- [1] [http://de.wikipedia.org/wiki/Ubiquitous\\_Computing](http://de.wikipedia.org/wiki/Ubiquitous_Computing) v. 19.1.2012
- [2] Mark Weiser: *The Computer for the 21st Century (1991)* <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>(u.a. in: Scientific American, 265)
- [3] (1984) John Gage and Bill Joy of Sun Microsystems, *"The network is the computer"* [http://en.wikipedia.org/wiki/John\\_Gage](http://en.wikipedia.org/wiki/John_Gage)
- [4] HJP Consulting, *GlobalTester*, <http://de.wikipedia.org/wiki/GlobalTester>
- [5] QR-Code, [http://en.wikipedia.org/wiki/QR\\_Code](http://en.wikipedia.org/wiki/QR_Code)
- [6] Studien des Büros für Technikfolgen-Abschätzung beim Deutschen Bundestag - 31, Michael Friedewald, Oliver Raabe, Pter Georgieff, Daniel J. Koch, Peter Neuhäusler: *Ubiquitäres Computing, Das „Internet der Dinge“ - Grundlagen, Anwendungen, Folgen*
- [7] Schoch/Strassner (2003): *„Wie smarte Dinge Prozesse unterstützen“* in: Sauerburger (Hrsg.): *Ubiquitous Computing*. Heidelberg, S. 23–31
- [8] *Next Generation Media* <http://www.nextgenerationmedia.de/>
- [9] Free Software Foundation, *Kurzzusammenfassung des Rechtsstreits*, <http://fsfe.org/projects/fff/avm-gpl-violation.de.html>
- [10] [http://de.wikipedia.org/wiki/Cloud\\_Computing](http://de.wikipedia.org/wiki/Cloud_Computing) Cloud Computing, Wikipedia.DE
- [11] K. Knopper, inaugural keynote "The next 100 years" at LinuxAsia/New Delhi 2006 <http://knopper.net/keynote-the-next-100-years-screen-linuxasia.pdf>
- [12] Kraiss KF: *„99% Langeweile und 1% panische Angst“* - über die Schwierigkeiten beim Umgang mit hochautomatisierten Systemen, sowie Günter Reichart, Leiter der Fahrzeug- und Verkehrsforschung bei BMW, 2000 zitiert auf <http://www.heise.de/ct/artikel/Mensch-denkt-Chip-lenkt-287572.html>
- [13] Anleitung des Chaos Computer Club zum Erzeugen eines Fingerkuppen-Reliefs <http://www.spiegel.de/video/video-28686.html>
- [14] Bilder übernommen von <http://de.wikipedia.org/wiki/RFID>
- [15] *„Phishing“*, <http://de.wikipedia.org/wiki/Phishing>
- [16] XBMC Open Source Media Player und Entertainment Center, <http://xbmc.org/about/>



# Visual Scripting: scripting the unscriptable

Wilhelm Meier

w.meier@unix.net

<http://mozart.informatik.fh-kl.de>

*Visual Scripting* ist eine neue Art, Applikationen und Systeme skriptbasiert zu steuern. Auch wenn eine Applikation kein Skript-API dafür anbietet, wird mit Hilfe von *Visual Scripting* die Möglichkeit geschaffen, jede beliebige Applikation, die ein GUI anbietet, über ECMA-Script zu steuern. Dies wird erreicht, indem die graphische Ausgabe der Applikation mit Hilfe von Bildverarbeitung und Mustererkennung untersucht wird. Zur Steuerung werden dann Tastatur- und Maus-Ereignisse für die Applikation synthetisiert. Anwendungsbeispiele hierfür sind Abläufe in legacy-Applikationen, Desktop-Automatisierung oder auch Systemverwaltung.

## 1 Einführung

Die meisten Unix/Linux-Benutzer und vor allem natürlich Systemadministratoren sind es gewohnt, wiederkehrende Aufgaben als Shell-Skripte zu automatisieren. Dies funktioniert deswegen so gut, weil Systemverwaltungskommandos vollständig über Parameter der Kommandozeile zu steuern sind und die Shell den notwendigen Satz an Kontrollstrukturen für den Skriptablauf mitbringt.

Leider ist die Situation bei Programmen mit GUI oftmals nicht so komfortabel: nicht jedes Programm stellt ein Script-API oder eine eigene Makro-Sprache zur Verfügung. Manchmal unterstützt das API auch nicht alle Kommandos oder Möglichkeiten, die über das GUI bereitstehen. Wiederkehrende Aufgaben können dann nicht effizient ohne Benutzerinteraktion automatisiert werden.

In diesem Fall wäre es hilfreich, wenn man einen Automat hätte, der diesselbe Verarbeitung wie ein tatsächlicher Benutzer leistet: Analyse des Programmzustandes *allein* anhand des GUI und Senden von Tastatur-, Maus- oder anderen HID-Ereignissen *allein* über das Fenstersystem [1]. In diesem Papier wird genau solch ein Automat beschrieben (im folgenden `qvscript` genannt). Dieser ist in Qt / C++ [2] realisiert und bietet die Möglichkeit, graphische Inhalte in einem Applikationsfenster, des gesamten Desktop oder von entfernten Applikationen via VNC, RDP oder X11 zu analysieren und in ECMA-Script[3] geschriebene Skripte ablaufen zu lassen.

Die Analyse des Fensterinhaltes findet mit Hilfe von Mustererkennung und Bildverarbeitungsverfahren (OCR) [5] [6] [7] statt. Scripte werden in ECMA-Script erstellt und `qvscript` bietet eine Laufzeit-Umgebung zur Analyse der graphischen Ausgabe und zur Synthetisierung von Eingabe-Ereignissen. Um effizient Skripte entwickeln

zu können, hat qvscript Möglichkeiten, zu erkennende Inhalte (Bildmuster) graphisch zu definieren, Skripte zu editieren und zu testen bzw. in einem Debugger laufen zu lassen. Weiterhin bietet qvscript selbst wiederum die Möglichkeit, beispielsweise per shell-Skript über eine socket- oder DBUS-Schnittstelle gesteuert zu werden. Damit ergeben sich interessante weitere Anwendungsfelder wie bspw. die Abbildung einer GUI einer nicht skriptfähigen Applikation auf eine neue Art der Benutzerschnittstelle.

## 2 Ansatz des *Visual Scripting*

Die Idee von *Visual Scripting* besteht darin, dem Anwender die Möglichkeit zu geben, eine skript-agnostische Applikation durch eine graphische GUI-Analyse zu steuern. Ohne das API von KDE zu kennen, kann doch jeder Benutzer das Startmenu von KDE durch einen entsprechenden Mausklick aktivieren. *Visual Scripting* realisiert diese Ak-

tion bspw. durch folgende Skriptzeile: `desktop.click();`. Hier wird in der Bildquelle `desktop` nahe dem Bildmuster  gesucht und anschließend ein Mausklick darauf ausgelöst.

Sollte das Startmenu noch nicht angezeigt werden, so kann erst die Maus an den unteren Bildschirmrand positioniert werden: `desktop.mouseMove(0, 800);` Eine *Drag-*

*Drop*-Aktion wird entsprechend mit `desktop.mouseDragDrop(testDatei.txt, Erster);`

dargestellt. Dabei findet ein *Drag* von  auf das Icon  statt.

Weitere Aktionen wie die Eingabe von Text, das Erkennen von Texten in bestimmten Regionen in Bezug auf ein erkanntes Bildmuster oder auch die Einschränkung der Suche auf bestimmte Regionen sind möglich. Zusätzlich können auch mit Hilfe von externer OCR-Software (z.B. *GOOCR* [5] oder *CuneiForm* [7]) Beschriftungen oder sonstige Texte erkannt werden und in der Skriptprogrammierung verwendet werden.

Um visuelle Skripte zu erstellen, sind vom Benutzer die notwendigen Icons, Buttons oder sonstige GUI-Elemente zu identifizieren und in einem qvscript-eigenen Repository als Bildmuster abzulegen. Anschließend können dann die Skripte in *ECMA-Script* erstellt werden, wobei auf das Script-API in der oben beispielhaft genannten Art zugegriffen wird.

Bei der Erstellung der Skripte wird der Benutzer durch einen speziellen Editor unterstützt (s.a. Abbildung 1).

## 3 Analyse durch Mustererkennung

Das Ziel von qvscript ist es, jede Applikation skriptfähig zu machen. Damit scheiden Ansätze aus, die ein bestimmtes API in der Applikation voraussetzen aus. Aber auch Methoden, die GUI-Elemente der Applikation über entsprechende Label im Fenstersystem identifizieren, sind nur bei kooperativen Applikationen möglich. Skript-agnostische Applikationen können nur durch eine graphische Analyse der Ausgabe gesteuert werden. Aus diesem Grund benutzt qvscript Verfahren der Mustererkennung, um Icons oder Buttons einer Applikation zu erkennen.

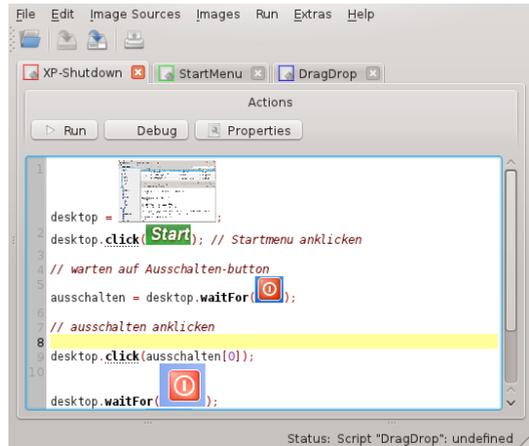


Abb. 1: Der Editor mit einem Skript zum Herunterfahren einer Windows-XP-Instanz

Im Gegensatz zur traditionellen

Bildanalyse, bei der Realweltszenen, die durch einen optischen Sensor aufgenommen wurden, analysiert und Objekte darin erkannt werden sollen, handelt es sich hier bei der Analyse von Bildschirmmustern um eine vergleichsweise einfache Aufgabenstellung. Daher reicht es hier aus, simple Erkennungsverfahren wie *Kreuzkorrelation* oder *relative Bilddifferenzen* zu verwenden. Dies hat den Vorteil eines geringen Rechen- und Zeitaufwandes zur Laufzeit, gleichzeitig aber auch den Nachteil einer geringen Robustheit gegenüber Toleranzen (Farbschemata, Skalierungen).

qvscript benutzt eine schnelle Form der Kreuzkorrelation, die nicht invariant gegenüber Skalierungen oder Farbänderungen ist, um Bildregionen zu erkennen. Es hat sich in der Praxis gezeigt, dass dieses Verfahren für die meisten Anwendungsfälle ausreichend ist. Über eine entsprechende Bildverarbeitungsbibliothek [8] können aber auch andere Erkennungsverfahren ausgewählt werden.

#### 4 Applikationsskripte

Abbildung 1 zeigt den Editor von qvscript zur Erstellung von Applikationsskripten. Der Editor bietet Syntaxhervorhebung sowie auch Code-Vervollständigung sowohl für ECMA-Script wie auch für das qvscript-API. Ein Script-Debugger ist ebenfalls enthalten. Bildquellen (ImageSource) und auch Bilder / Bildmuster (Image) des Repositories können hier visuell eingefügt werden. Dies erleichtert den Umgang mit den Skripten.

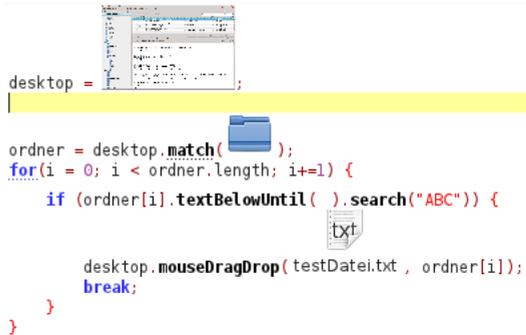
Objekte des Typs `ImageSource` unterstützen u.a. die folgenden Operationen (s.a. Abbildung 2):

- `match(image) → list of matches`: es werden alle Vorkommnisse eines Bildmusters `image` in der Bildquelle gesucht. (Beispiel: `listOfMatches = vnc.match ( Benutzername: );`)
- `click(match|matchList|image)`: es wird ein einfacher Mausklick auf den definierten Klickpunkt des `match`, des `image` oder der Elemente der `matchList` synthetisiert. (Beispiel: `vnc.click(listOfMatches[0]);`)
- `waitFor(image) → list of matches`: wie `match`, allerdings wird permanent nach `image` gesucht (bis zu einem per-Skript `timeout`)
- `doubleClick(match|matchList|image)`: wie `click`, nur als synthetisierter Doppelklick
- `mouseMove(match|(x,y))`: Positionierung des Mauszeigers
- `mouseShake()`: schnelle, kleine Mausbewegung relativ zur aktuellen Position
- `mouseDragDrop(match|matchList|image,match|matchList|image)`: Synthetisierung einer *drag-drop*-Mausoperation
- `type(match|matchList|image,"txt",...)`: Synthetisierung einer Tastatureingabe `txt` an der Position des `match` oder `image`
- `show(match|matchList|image)`: Darstellung der Positionen von `match`, ... als Overlay in der Bildquelle.

Der Datentyp `Match` bietet u.a. die Operationen an:

- `textBelowUntil(match|image|delta) → text`: es wird der Text darunter bis zu einem begrenzenden `match`, `image` oder eine Abstand `delta` erkannt. (Beispiel: `text = listOfMatches[0].textBelowUntil(20);`)
- `textAboveUntil(match|image|delta) → text`: analog zu `textBelowUntil()`;
- `textRightUntil(match|image|delta) → text`: analog zu `textBelowUntil()`;
- `textLeftUntil(match|image|delta) → text`: analog zu `textBelowUntil()`;

Weiterhin können Bereiche innerhalb einer Bildquelle mit Hilfe des Typs `Region` eingegrenzt werden: so kann beispielsweise aus einer Liste von Bildmustern eine umschreibende, rechteckige Region gebildet werden. Innerhalb eines Bereiches stehen dann die Suchoperationen wie bei `ImageSource` zur Verfügung.



```

desktop = ...;
ordner = desktop.match( ... );
for(i = 0; i < ordner.length; i+=1) {
    if (ordner[i].textBelowUntil( ).search("ABC")) {
        desktop.mouseDragDrop(testDatei.txt, ordner[i]);
        break;
    }
}

```

Abb. 2: Ein Skript zum Verschieben einer Datei `testDatei.txt` in den Ordner mit dem Namen `ABC`.

Über die Operationen `left` → `Region`, `right` → `Region` ... können Bereiche weiter eingegrenzt werden. Bei sehr großen Bildquellen mit großen Auflösungen kann dies die Suchgeschwindigkeit stark positiv beeinflussen. Zudem bietet dies die Möglichkeit, Mehrdeutigkeiten bei der Erkennung von Bildmustern zu vermeiden.

Die o.g. Operationen stellen nur einen Ausschnitt aus den Möglichkeiten dar. Eine vollständige Auflistung würde den Rahmen dieses Artikels überschreiten.

In `qvscrip`t können Bildquellen der Typen `Desktop`, `Foreign`, `Application`, `Window` und `VNC` definiert werden. `Desktop` bezeichnet den vollständigen lokalen Desktop und `Window` das lokale Fenster einer bestimmten Applikation (die Applikation kann anhand der Fenstermerkmale wie Titel, ... identifiziert werden). Der Typ `Foreign` ist eine Verbindung zu einem fremden Display via X11 und analog dazu der Typ `VNC` via VNC-Protokoll. Der Typ `Application` startet eine Anwendung in einem geschachtelten X11-Server

(Xephyr). Dies ist hilfreich, wenn Anwendungen zwar auf dem lokalen Rechner laufen sollen, ihre Fenster aber unsichtbar bleiben sollen, um die Arbeit am Desktop nicht zu behindern. Auf eine direkte Implementierung des RDP-Protokolls wie bei VNC wurde zunächst verzichtet (es wird allerdings daran gearbeitet). Stattdessen kann man den Typ `Application` verwenden und in dem geschachtelten X11-Server einen `rdesktop`-Client starten.

## 5 Anwendungen und Zusammenfassung

Die Anwendungsbereiche von `qvscrip`t sind sehr vielfältig: ein häufiger Einsatzzweck ist die Automation von wichtigen *legacy*-Applikationen, oft auch aus dem Windows-Umfeld, die kein externes Script-API anbieten [9]. Eine derartige Applikation kann bspw. über RDP direkt oder in einer `qemu`-VM-Instanz über VNC gesteuert werden, beispielsweise um einen periodischen Datenexport durchzuführen.

Andere Einsatzzwecke finden sich oft in der Systemadministration. Zur *end-to-end* Überwachung von Schulungsrechnern, die über ein Netzwerkbootssystem starten, kann eine *qemu*-VM-Instanz via *pxe* booten. Aus dem Bootmenu wird dann mittels *qvscript* der passende Eintrag ausgewählt bis hin zum graphischen Login eines Testbenutzers. Bei erfolgreichem *Login* wird von *qvscript* eine E-Mail an eine Überwachungsinstanz gesendet oder bspw. ein *Nagios*-Plugin aktiviert. Ob die Anmeldung fehlerlos funktionierte, wird durch Icons auf dem Desktop des Testbenutzers durch das Script erkannt.

Mit *qvscript* wird ein Werkzeug vorgestellt, mit dem *beliebige* Applikationen skriptgesteuert werden können. Dabei ist es möglich, dass diese lokal oder entfernt ablaufen und per *X11*, *VNC* oder *RDP* zugreifbar sind. Zur Analyse der Applikationsoberfläche stehen verschiedene Mustererkennungsverfahren zur Verfügung. Einen vergleichbaren Ansatz stellt das Projekt *Sikuli* [4] dar. Die Implementierung findet hier in *Java* statt, visuelle Skripte werden in *Jython* geschrieben. Allerdings bietet *Sikuli* keine Unterstützung für *remote-X11*, *VNC* oder *RDP*. Durch den Einsatz einer *C++*-Bildverarbeitungsbibliothek sind die Laufzeitergebnisse vergleichbar.

## Literatur

- [1] Sissel, J.: *xdotool: fake keyboard/mouse input, window management and more*. Mai 2011, <http://www.semicomplete.com/projects/xdotool>
- [2] *A cross-platform application and UI framework*. Nokia, Nov. 2011 <http://qt.nokia.com/products>
- [3] *Standard ECMA-262: ECMAScript Language Specification*. ECMA-International, Jun. 2011 <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [4] *A visual technology to automate and test graphical user interfaces*. Project Sikuli, Dez. 2010, <http://sikuli.org>
- [5] Schulenburg, J.: *GOOCR*, Sep. 2010, <http://jocr.sourceforge.net>
- [6] *Tesseract OCR engine*, Dez. 2011, <http://code.google.com/p/tesseract-ocr>
- [7] *CuneiForm OCR engine, Linux port*, Dez. 2011, <https://launchpad.net/cuneiform-linux>
- [8] *The CImg Library: C++ Template image processing toolkit*, Dez. 2011, <http://cimg.sourceforge.net>
- [9] *KMUX - Linux und F/OSS für kleine und mittelständische Unternehmen*, Apr. 2011, <http://www.kmux.de>

# Zero Commercial Software Strategy - Eine Fallstudie

**Frederik Kramer und Naoum Jamous**  
{frederik.kramer, naoum.jamous}@ovgu.de  
<http://www.mrcc.eu>

Unter anderem auf Grund des Mangels an stringenter Planung können beim Einsatz von Open Source Software (OSS) in kleinen und mittleren Unternehmen (KMU) ähnliche Probleme auftreten, wie auch beim Einsatz proprietärer Software. Anstatt die Vor- und Nachteile des Einsatzes von Open Source Software anhand bestimmter Anwendungssysteme darzustellen, verfolgt dieser Beitrag ein Extremziel. Am Beispiel eines kleinen IT-Dienstleisters geht der vorliegende Artikel der Frage nach, ob eine reine Open Source Strategie prinzipiell umsetzbar ist und ob diese unter quantitativen und qualitativen Gesichtspunkten sinnvoll sein kann.

## 1 Einführung in die Fallstudie

Das im Rahmen dieser Fallstudie dargestellte Unternehmen ist ein kleiner IT-Dienstleister mit Sitz in Rosengarten bei Hamburg. Es beschäftigt derzeit neben den beiden Geschäftsführern vier weitere Softwareentwickler und Consultants.

Es wurde für dieses Forschungsvorhaben aus verschiedenen Gründen gewählt. Zum einen ist einer der Autoren dieses Beitrags am betreffenden Unternehmen beteiligt und konnte daher eine entsprechenden Strategievorschlag selbst einbringen. Zum anderen verfügen die Autoren aus diesem Grund über Zugang zu Unterlagen und Erkenntnissen, die während der mehrere Jahre andauernden Strategieumsetzung angefallen sind. Es wird zudem davon ausgegangen, dass kein anderes Unternehmen die unkonventionelle, wenn nicht sogar sinnlos anmutende Strategie des Komplettverzichts auf proprietäre Software erproben bzw. nachhaltig verfolgen würde.

Auf Grund des Mangels an Alternativen sowie der langjährigen Erfahrungen, die das Unternehmen beim Einsatz von Open Source Software und bei deren Nutzung auch in Verbindung mit proprietären Produkten bereits zum Zeitpunkt der Strategiefestlegung gesammelt hatte, tritt aus Sicht der Autoren die häufig beschriebene Voreingenommenheit des Forschenden in der Fallstudienforschung in den Hintergrund [vgl. hierzu 1, 2, 5].

Allerdings erhebt der Beitrag deshalb explizit nicht den Anspruch der Allgemeingültigkeit. Ziel ist es vielmehr, die während der Strategieumsetzung erlangten Erkenntnisse, sowie die zum Einsatz gekommenen Open Source Komponenten einer breiten, am Einsatz von OSS interessierten Öffentlichkeit zugänglich zu machen und mit dieser zu diskutieren.

## 2 Die Besonderheit und der Status von OSS

Die Zeiten des großen Open Source Hypes sind vorbei. Der von Linus Torvalds erfundene Entwicklungsprozess war Anfang der 1990er Jahre revolutionär, weil er mehrere, der damals geltenden Dogmen der Informationstechnologiebranche in Frage stellte.<sup>1</sup>

Allein der Linux Kernel ist seit seinem ersten offiziellen Release im September 1991 von 10.000 auf aktuell knapp 1.5 Millionen Codezeilen gewachsen [vgl. 4]. Das OSS Paradigma wurde in der Folge auf viele weitere Anwendungssystemtypen angewendet. Das bekannteste Softwarerepository „Sourceforge“ führt aktuell über 300.000 unterschiedliche Open Source Projekte<sup>2</sup>. In einigen Bereichen ist OSS heute weitgehend funktionsäquivalent mit proprietärer Software bzw. sogar führend<sup>3</sup>.

Im Zusammenhang mit dem großen Erfolg von OSS drängte sich den Autoren dieses Artikels deshalb die Frage auf, ob auf proprietäre Software überhaupt vollständig verzichtet werden kann und ob dieser Totalverzicht aus quantitativen bzw. qualitativen Gesichtspunkten zudem sinnvoll sein kann. Diesen beiden Teilfragen geht dieser Beitrag deshalb nach.

## 3 Die „Zero Commercial Software Strategy“

Bereits zu Beginn des Betrachtungszeitpunktes im Jahr 2007 hatte das Unternehmen einige Jahre Erfahrung mit OSS gesammelt. Um Missverständnissen der Leser bereits an dieser Stelle vorzubeugen, sei gesagt, dass die Autoren bei der Strategieentwicklung weder davon ausgegangen sind noch davon überzeugt waren, dass sich OSS immer und in jedem Umfeld einsetzen lässt. Vielmehr sind die Autoren der Auffassung, dass der sinnvolle Einsatz von OSS nur auf Basis solider hybrider Bewertungsmodelle erfolgen kann. Diese Auffassung wird unter anderem auch durch die Erkenntnisse einer in diesem Zusammenhang angefertigten Diplomarbeit bestärkt [vgl. hierzu 3].

Die „Zero Commercial Software Strategy“ ist deshalb als Strategieexperiment konzipiert. Sie hat nicht zum Ziel, den bedingungslosen Einsatz von OSS zu empfehlen, sondern alle wertschöpfenden Aktivitäten innerhalb des Unternehmens mit Hilfe von OSS abzubilden, um die prinzipielle Machbarkeit dieser Extremstrategie zu prüfen und dabei etwaige qualitative und quantitative Risiken aufzudecken.

Zunächst muss in diesem Zusammenhang erläutert werden in welchen Bereichen das betreffende Unternehmen wertschöpfend tätig ist. Abbildung 2 zeigt das Portfolio des Unternehmens, das im Wesentlichen aus IT-Dienstleistungen in den Bereichen **Software Engineering, Communications** und **Consulting** besteht.

<sup>1</sup>Offenlegung des Quellcodes sowie Herstellung von Software durch ein Heer Freiwilliger

<sup>2</sup>siehe hierzu <http://sourceforge.net/about>, zuletzt besucht am 17.1.2012

<sup>3</sup>exemplarisch seien hier der Webserver *Apache* und das Content Management System *Typo3* genannt.

Die Abbildung zeigt darüber hinaus, welche Aktivitäten in diesen drei Bereichen mit entsprechender Anwendungssoftware unterstützt werden muss. Die Zuordnung der Aktivitäten zu Wertschöpfungsbereichen ist nicht scharf abgrenzbar. So werden zum Beispiel **Office Tools** nicht ausschließlich für den Bereich Consulting benötigt, sondern auch für die nicht explizit aufgeführte Administration des Unternehmens und zu einem kleineren Teil auch für das **Software Engineering** und die Dienstleistungen im Bereich **Communications**.

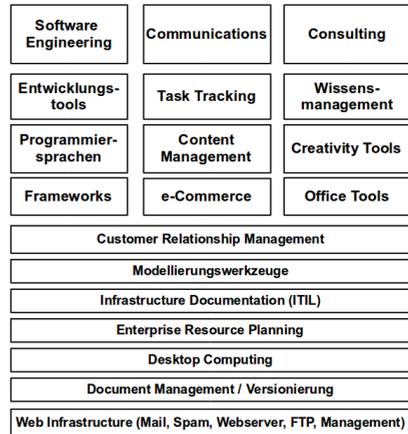


Abbildung 2: Wertschöpfungsaktivitäten

Horizontal bereichsübergreifend werden jedoch nur Anwendungssysteme benötigt, die alle drei wesentlichen Wertschöpfungsbereiche unterstützen. Dies sind **Customer Relationship Management**, **Modellierungswerkzeuge**, **Infrastructure Documentation** und **Enterprise Resource Planning**, die zusammen die wesentlichen Schnittstellen zu den Kunden bilden.

## 4 Strategieumsetzung und -bilanzierung

Der folgende Abschnitt beschreibt den Ablauf der Strategieumsetzung. Er ist in die Teilbereiche *Beginn der Strategieimplementierung* (siehe Abschnitt 4.1), *Integration und Customizing* (siehe Abschnitt 4.2) sowie die *ökonomische Bilanzierung* (Abschnitt 4.3) und *qualitative Bilanzierung* (siehe Abschnitt 4.4) unterteilt.

### 4.1 Beginn der Strategieimplementierung

Zu Beginn der Strategieimplementierung verfügte das Unternehmen noch nicht über ein klar strukturiertes Wertschöpfungsportfolio wie in Abbildung 2 dargestellt. Der Fokus lag 2007 noch auf der umfassenden Beratung von Kleinunternehmen und Einzelunternehmern. Wesentlicher Geschäftszweig war zu diesem Zeitpunkt Hardwaregeschäft.

Daneben bestand bereits, der vor allem auf den Onlinemarkt ausgerichtete Bereich **Software Engineering**. Software Engineering wurde allerdings komplementär zum Hardwaregeschäft verstanden und angeboten. Das integrierte Lösungskonzept im Sinne einer umfassenden IT-Beratung war jedoch bereits damals tragende Wertschöpfungsaktivität des Unternehmens.

Insbesondere in diesem Bereich zeigte sich, dass der kontinuierlich wachsende Erfahrungsschatz der Mitarbeiter bei der Entwicklung kundenspezifischer Lösungen sowohl Wissensmanagement als auch eine strukturierte Koordination von Teilaufgaben und Aufgabenträgern erforderten. Da die Mitarbeiter des Unternehmens schon damals über zwei (heute drei) Bundesländer verteilt waren, war zudem eine flexible möglichst webbasierte Lösung notwendig.

## 4.2 Integration und Customizing

Zwar hatte das Unternehmen bereits Ende 2004 die proprietäre Warenwirtschaftslösung Lexware Faktura durch das zu diesem Zeitpunkt quelloffene CAO-Faktura abgelöst, allerdings genügte dieses Anwendungssystem schon bald den wachsenden Anforderungen nicht mehr. Zum einen wurde der Quellcode von CAO-Faktura nicht mehr weiter veröffentlicht. Zum anderen genügte der verwendete Softwarestack um Borlands Delphi, der oben beschriebenen Strategie nicht.

Es wurde also unter Erweiterung der Methodik aus der in 2007 durchgeführten Diplomarbeit [vgl. 3] eine mittelfristige Ablösung durch die vollständige quelloffene Enterprise Resource Planning Lösung OpenERP beschlossen. Diese sollte wie für Software diesen Typs üblicherweise gefordert, als zentrales Element der Unternehmens-IT, alle Querschnittsprozesse (z.B. Beschaffung, Stammdatenmanagement, Rechnungswesen und Controlling) abbilden.

Für das bereits beschriebene Task- und Wissensmanagement kamen das Bugtracking-tool Mantis und das Wikisystem Docuwiki zum Einsatz. Die operative Koordination von Aktivitäten und Ressourcen erfolgt bereits seit Mitte des Jahres 2009 mit Hilfe dieser Anwendungssystemkombination. Eine weitreichende Integration mit OpenERP zur Dokumentation des Leistungserstellungsprozesses und Erstellung der Spe-  
senabrechnung erfolgte nach dessen Einführung Anfang 2011.

Auf Ebene der Infrastruktur kommen neben den relationalen Datenbanksystemen MySQL und PostgreSQL, Ubuntu Linux als Betriebssystem sowie KVM als Hypervisor zum Einsatz. Der Hypervisor ermöglicht die Abstraktion der physikalischen Hardware auf mehrere virtuelle Nodes, die wiederum die Plattform für Anwendungssysteme auf höheren Ebenen bilden.

Für die Bereitstellung der Webinfrastruktur (siehe Abbildung 2) kommen neben dem Webserver Management Tool ISPConfig, Apache, Amavis, Dovecot, ClamAV und ProFTPd zum Einsatz. Die Anwendungsentwicklung erfolgt unter Nutzung verschiedener Open Source Versionierungssysteme (SVN, Mercurial und Git) sowie Frameworks (Yii, Smarty). Zur Bereitstellung von Content Management kommen je nach Anwendungsfall entweder Joomla, Typo3 oder Wordpress zum Einsatz. Die Grundlage für e-Commerce bilden Oxid und Magento. An der Integration von Oxid mit OpenERP wird zum Erstellungszeitpunkt des Artikels gearbeitet.

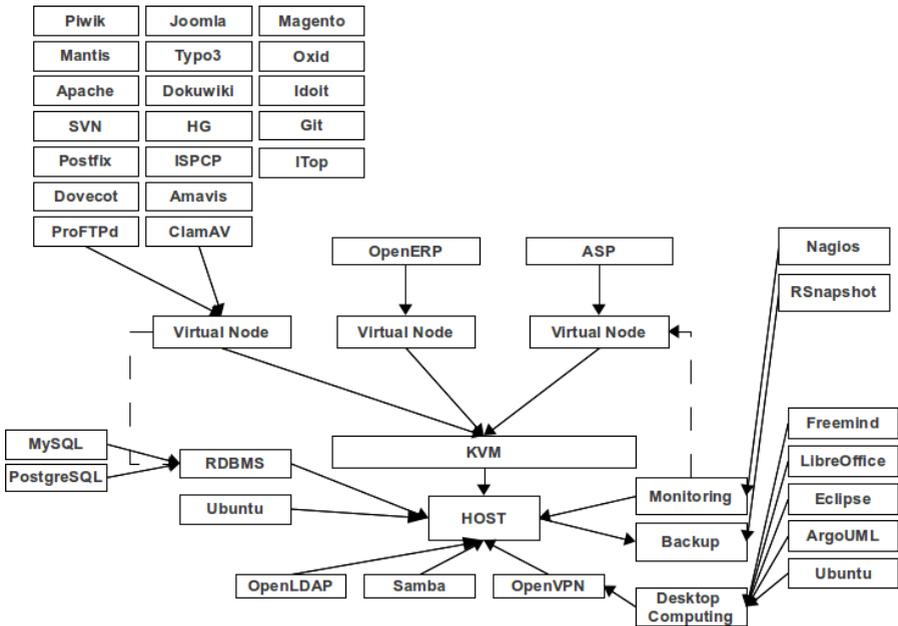


Abbildung 3: Architektur

Das Backup und Überwachungskonzept basiert ebenfalls vollständig auf Open Source Software. Hier kommen Nagios zum Monitoring und RSnapshot zum Backup der Filesysteme zum Einsatz. Weitere genutzte Dienste sind Samba für das Bereitstellen von Dokumenten und OpenVPN zur Zugangssicherung. Das Desktop Computing bei den Mitarbeitern erfolgt wie auch der Serverbetrieb auf Basis von Ubuntu Linux. Die Desktops und Notebooks der Mitarbeiter sind per OpenVPN an die zentrale Infrastruktur angebunden und verfügen zudem über ArgoUML, Eclipse, LibreOffice sowie Freemind und diverse Clients für die Versionierungssysteme.

Die Umsetzung der Strategie bzw. insbesondere die Integration der verschiedenen Anwendungssysteme erfolgte über mehrere Zwischenschritte über eine Gesamtdauer von nahezu vier Jahren. Dies resultiert aus einem gemischten Strategieimplementierungsansatz. Einerseits wurden je nach Bedarf neue Anwendungssysteme auf höherer Ebene (top-down) erprobt und unmittelbar eingesetzt (z.B. Joomla, Piwik, Magento, Idoit u.ä.) andererseits wurde erst gegen 2009 die gesamte Infrastruktur von mehreren physikalischen Systemen, damals noch unter Verwendung des Hypervisors XEN, konsolidiert (bottom-up). Gegen Mitte 2010 erfolgten schließlich die Migration auf KVM sowie verschiedener virtueller Hosts von Debian auf Ubuntu 10.04 LTS.

Den Auswahlprozess jedes einzelnen Anwendungssystems ausführlich darzustellen

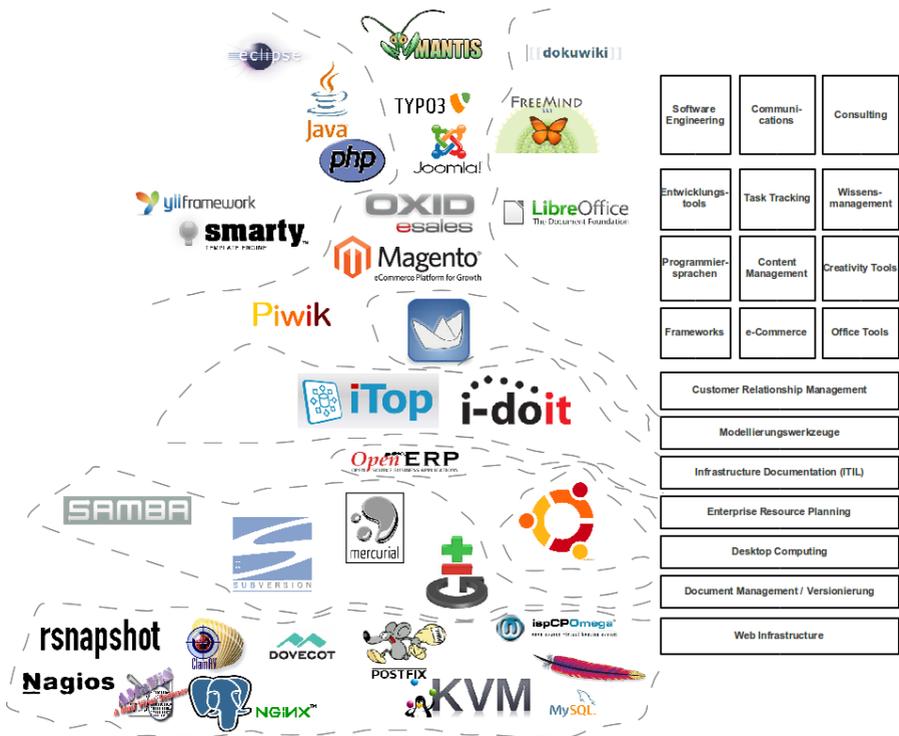


Abbildung 4: Zuordnung von Komponenten zu Wertschöpfungsaktivitäten

len, würde den Rahmen dieses Beitrags sprengen. Stattdessen sei auf die Abbildungen 3 und 4 verwiesen, die zum einen die Architektur des durch konsequente Strategieverfolgung entstandenen betrieblichen Informationssystems als Ganzes darstellt (Abbildung 3), zum anderen aber auch die Zuordnung der einzelnen Komponenten zu den zuvor dargestellten wertschöpfungsrelevanten Aktivitäten (Abbildung 4<sup>4</sup>) vornimmt.

### 4.3 Ökonomische Bilanzierung

Der Begriff **Strategie** besitzt keine eindeutige Definition. Im Volksmund wird häufig von einem langfristigen, bewußt geplanten und abstrakten Maßnahmenbündel ausgegangen. Mintzberg bezeichnet jedoch in seinem vielzitierten Werk den Strategiebegriff ebenso als *spontanes Handeln* und *unbewußt entstehendes Handlungsmuster* [vgl. 6,

<sup>4</sup>die dargestellten Logos sind geistiges Eigentum der jeweiligen Projekte und dürfen zu kommerziellen Zwecken ohne Erlaubnis der Rechteinhaber nicht verwendet werden.

S.]. Vor allem die letzte Begriffsdefinition passt besser zum betrachteten Fallbeispiel als die Vorstellung von einem langfristigen geplanten Maßnahmenbündel. Letzteres entspricht darüber hinaus nicht der eher operativen Ausrichtung vieler KMU. Den Erfolg einer Strategie an monetären Einheiten zu messen, ist schon wegen des unklaren Wirkungszusammenhangs nicht zielführend. Was im Zusammenhang mit der Umsetzung der oben beschriebenen Strategie jedoch eindeutig festzustellen war, ist, dass der Einsatz von OSS, obgleich er häufig zunächst sehr einfach erscheint, im Detail ressourcenhungrig und damit auch kostenintensiv sein kann.

Zwar ist die reine Installation von z.B. Joomla in wenigen Minuten erledigt, aber alleine das automatisierte Update vieler JoomlaInstanzen erfordert Zusatzentwicklung. Für den Betrieb mehrere Domains mit nur einer JoomlaInstanz<sup>5</sup> fällt ebenso Zusatzaufwand an. Um das Management von Webinfrastrukturen wiederum für viele Kunden sicherzustellen und ihnen dabei gleichzeitig die Möglichkeit zu eröffnen, gewisse Änderungen selbst durchzuführen (z.B. Anlegen neuer Email-, und FTP-Accounts), bedarf es eines Äquivalents zu kommerziellen Tools wie etwa Plesk. Hier hat es ISPCP zu beachtlicher Reife geschafft. Die funktionalen Unzulänglichkeiten früherer Versionen zu umgehen, erforderte jedoch durchaus erfahrenes Personal, nicht unerheblich Zeit und damit auch Geld.

Den Gesamtaufwand der Umsetzung der oben genannten Strategie zu bestimmen, ist aus Sicht der Autoren nicht zielführend. Es kann jedoch gesagt werden, dass die Integration verschiedener Open Source Anwendungssysteme, obwohl sie im Detail schwierig und aufwendig sein mag und zum Teil sogar ist, oft ohne große architektonische Hürden erfolgen kann. Die Updatesicherheit der Open Source Anwendungen bleibt damit i.d.R. erhalten. Viele der im konkreten Fall eingesetzten Anwendungssysteme verfügen mittlerweile zudem über sehr ausgereifte APIs und Pluginsysteme (z.B. Joomla, OpenERP, Magento, Oxid, ISPCP) und haben eine beachtliche Reifung hinter sich.

Zumindest im Vergleich mit dem Aufwand, den das Unternehmen betreibt, um die vielen Windows-basierten Anwendungssysteme seiner Kunden zu administrieren und an geänderte Bedürfnisse der Kunden anzupassen, fällt der Wartungsaufwand und Anpassungsaufwand des eigenen Informationssystems gering aus.

#### 4.4 Qualitative Bilanzierung

Eine qualitative Bilanzierung lässt sich einfacher aufstellen, als die ökonomische Gegenüberstellung. Nach rund vier Jahren Dauer verfügt das betrachtete Unternehmen spätestens seit der vollständigen Ablösung von CAO-Faktura durch OpenERP über eine betriebliches Informationssystem, das alle Wertschöpfungsaktivitäten vollständig mit Open Source Software abbildet. Lediglich einige Mitarbeiter in der Verwaltung bevorzugen gewohnheitshalber Windows Betriebssysteme. Die Integration mit den sonstigen Infrastrukturkomponenten stellt jedoch in diesem Fall kein Hindernis

<sup>5</sup>bei gleichzeitiger suchmaschinenoptimaler Trennung der Pfade mittels Apache `mod_rewrite`

dar<sup>6</sup>. Da es sich beim Einsatz von Windows um keine harte Abhängigkeit handelt, kann deshalb trotzdem von einer Totalimplementierung der Strategie gesprochen werden.

Die Integration mit neuen Komponenten (wie etwa kürzlich Piwik) erfolgt mittlerweile sehr routiniert und weitgehend nahtlos. Natürlich muss in diesem Zusammenhang gesagt werden, dass während einer über mehrere Jahre andauernden Strategieimplementierung auch Kenntnisse und Fertigkeiten im Umgang mit Technologien gewonnen und vertieft werden, die die notwendigen Aufgaben beschleunigen. Dies würde jedoch auch für ein geschlossenes Produktportfolio bzw. eine Closed Source Strategie zumindest bedingt gelten.

## 5 Zusammenfassung und Ausblick

Zusammenfassend kann zumindest für den hier dargestellten Fall gesagt werden, dass eine „Zero Commercial Software Strategy“ definitiv umsetzbar ist. Das heißt im Umkehrschluss, dass das früher häufig geäußerte Argument, für einige Anwendungsgebiete existiere keine geeignete OSS, zumindest für den IT-Dienstleistungsbereich, widerlegt werden konnte. Aus den Erkenntnissen kann man jedoch nicht schließen, dass eine ähnliche Strategie zum Beispiel für einen mittelständischen Steuerberater umsetzbar ist. Die Autoren dieses Artikels gehen im Gegenteil sogar davon aus, dass selbst die prinzipielle Machbarkeit in vielen Fällen mit einem großen Fragezeichen versehen werden muss.

Die Abhängigkeit von der Branchenlösung Datev dürfte zum Beispiel derart groß sein, dass eine illusorisch große Zahl Steuerberatungskanzleien über einen längeren Zeitraum in die Entwicklung einer gemeinsamen Open Source Alternative investieren müssten, um in die Nähe einer prinzipiellen Umsetzbarkeit zu gelangen. Weil jedoch die wenigsten Steuerberater über die damit verbundene Risikobereitschaft in einem für sie unbekanntem Betätigungsfeld<sup>7</sup> verfügen dürften bzw. verfügen wollen, halten die Autoren eine ähnliche Strategie in vielen anderen als der IT-Industrie für schwierig bis gar nicht umsetzbar.

Nur dann, wenn die derzeitigen Marktführer in der jeweiligen Branche den Bogen aus einseitiger Abhängigkeit und Preispolitik überspannen, dürfte eine für die Umsetzung einer Totalstrategie notwendige, kritische Maße an Investition erreicht werden und auch nur dann wäre sie eine zielführende strategische Alternative.

Der hier vorgestellte Fall ist ein Extrembeispiel. Dennoch bietet er aus Sicht der Autoren einen Einblick in das prinzipiell machbare. Es konnte gezeigt werden, dass unter bestimmten Rahmenbedingungen ganz auf proprietäre Software verzichtet werden kann. Der Artikel regt zur weiteren qualitativen, wie auch gestaltungsorientierten Auseinandersetzung in diesem Forschungsgebiet an. Es bleibt zudem zu hoffen,

<sup>6</sup>weil alle Anwendungen des Desktop Computing (siehe Abbildung 3) auch in Windows Versionen verfügbar sind.

<sup>7</sup>gemeint ist die Informationstechnologiebranche

dass die Beliebtheit von OSS und die Bereitschaft vieler freiwilliger und institutioneller Contributor ungebrochen bleiben.

## Literatur

- [1] Izak Benbasat, David K. Goldstein, and Melissa Mead. The case research strategy in studies of information systems. *MIS Quarterly*, 11:369–386, September 1987.
- [2] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative Inquiry*, pages 219–245, 2006.
- [3] Frederik Kramer. Entscheidungsmodell für die migration / substitution konventioneller kommerzieller anwendungssysteme durch open source software. Master's thesis, Otto-von-Guericke-Universität Magdeburg, September 2007.
- [4] Thorsten Leemhuis. Kernel-log: 15.000.000 zeilen, 3.0 wird longterm-kernel. Website, January 2012. last visited 20.01.2012.
- [5] Judy McKay and Peter Marshall. Quality and rigour of action research in information systems. In *ECIS 2000 Proceedings*, 2000.
- [6] Henry Mintzberg. Five ps for strategy. *California Management Review*, 30(1):11–24, 1987.



## 3 Zusammenfassungen der weiteren Vorträge

### 3.1 Adaptive tickless kernel

*Frederic Weisbecker, Red Hat, Inc., fweisbec@gmail.com*

The scheduler tick is a timer that runs periodically to maintain the progression of time, scheduler state, rcu state, load statistics, timers, etc ... With 2.6.21 a new feature called dynticks came that shuts down the periodic tick when a CPU goes idle to reduce CPU energy consumption. Today we want to minimize the tick further when the CPU is busy running tasks. The goal is to reduce the critical sections that induce latencies. Throughput may also get better without useless interrupts to handle and the CPU cache trashing they cause.

### 3.2 Aktuelle Entwicklungen beim Linux-Kernel

*Thorsten Leemhuis, Heise Zeitschriften Verlag GmbH & Co. KG, thl@ct.de*

Der Vortrag gibt einen Überblick über die jüngsten Verbesserungen des Linux-Kernels. Diese sind vielfach auch für Allwerwelts-PCs von Belang und erreichen mit Distributionen wie Ubuntu 12.04 LTS oder Fedora 17 in Kürze eine breite Anwenderschar. Im Rahmen des Vortrags kommen auch die Neuerungen bei kernelnaher Software sowie einige noch in Entwicklung befindliche Änderungen am Kernel zur Sprache. Zudem werden der Entwicklungsprozess, das Versionsschema sowie einige andere für die kurz- und langfristige Entwicklung von Linux und Linux-Distributionen wichtige Themen angerissen.

Weitere Informationen: <http://www.heise.de/open/kernel-log-3007.html>

### 3.3 APT – Paketverwaltung für Debian GNU-Linux

*Thomas Winde, Thomas Winde Ausflugsfahrten, ausflug@web.de*

Jeder Debiannutzer hat wohl schon mal etwas vom Kommando «apt-get» gehört und wahrscheinlich auch schon dessen Parameter «update», «upgrade» und «install» benutzt. Das Apt-Paketverwaltungssystem ist aber eine viel umfangreichere Werkzeugammlung, mit der sich fast das komplette System verändern lässt und jede Menge Informationen über Debianpakete in und außerhalb des Systems zu finden sind.

### 3.4 Back to Unix: 100 POSIX-Kommandos statt 29.000 Packages

*Martin Neitzel, Gaertner Datensysteme, neitzel@gaertner.de*

Dieser Vortrag will dazu einladen, sich lieber mit dem klassischen Unix-Werkzeugkasten anzufreunden, als das Heil in vorgefertigten Spezialpaketen zu suchen. Vorgestellt werden die Unix-Kommandos des POSIX-Standards und der Umgang mit ihnen.

Weitere Informationen: <http://www.gaertner.de/neitzel/posix/>

### 3.5 Bash the Publisher – Vom Sinn und Unsinn des Verlagswesens

*Markus Wirtz, Open Source Press GmbH, info@opensourcepress.de*

Der Vortrag soll sich zu einer Diskussionsrunde entwickeln: Ausgehend von den Erfahrungen eines kleinen Computerbuchverlags geht es um die (Un)Möglichkeit von Dokumentation in Zeiten des Web 2.x. Ist Verlagsarbeit noch zeitgemäß? Was ist «Content»? Was bedeutet der Freiheitsbegriff der FOSS-Bewegung für gedruckte und digitale Bücher? Leserinnen und Leser von Fachbüchern sind herzlich eingeladen, Einblick in die Entwicklung eines Verlagsprogramms zu nehmen. Wer mit dem Gedanken spielt, selbst ein Buch zu schreiben, findet hier Gelegenheit, Veröffentlichungsoptionen miteinander zu vergleichen.

### 3.6 Best Practice OTRS – IT Service Management ganz praktisch

*Rico Barth, c.a.p.e. IT GmbH, info@cape-it.de*

Der Vortrag betrachtet die Projektpraxis bei der Einführung von IT-Service-Management-Prozessen (ITSM) auf Basis von OTRS. Ausgehend von der Erläuterung wichtiger Anforderungsaspekte vor dem Projekt wird auf die Toolauswahl vor der Realisierung und die Integration anderer Open-Source-Systeme eingegangen. Basierend auf Projekterfahrungen beleuchtet der Vortrag in realistischer Weise die praktischen Herausforderungen während solcher Einführungsprojekte. Dem Publikum werden mit dem Vortrag Hinweise und Ratschläge für eigene ITSM-Projekte in die Hand gegeben.

Weitere Informationen: <http://www.cape-it.de>

### 3.7 Conkeror und andere tastaturfokussierte Webbrowser

*Axel Beckert, ETH Zürich, abe@debian.org*

Fast jeder kennt und nutzt heute Firefox oder einen der proprietären Browser MSIE, Opera, Safari oder Google Chrome. Dennoch haben all diese Browser eine Sache gemeinsam: Sie sind auf Benutzung mit der Maus ausgelegt. Es gibt jedoch mittlerweile eine ganze Menge weniger bekannte Webbrowser, die konsequent auf Tastaturbedienung ausgelegt sind. Dieser Vortrag stellt einige davon vor.

Weitere Informationen: <http://noone.org/talks/conkeror/>

## 3.8 Continuous Integration mit Jenkins für Perl-Projekte

*Renée Bäcker, Perl-Services.de Renée Bäcker*

Nichts ist schlimmer, als eine fehlerhafte Software an Kunden auszuliefern. Deshalb sollten Tests und das Zusammenbauen der unterschiedlichen Komponenten regelmäßig auf dem Programm stehen. Wir setzen mittlerweile sehr erfolgreich Jenkins für die Tests ein. Der Vortrag soll zeigen, wie man Jenkins bei Perl-Projekten einsetzen kann.

## 3.9 CouchDB und CouchApps

*Roman Geber, B1 Systems GmbH, info@b1-systems.de*

CouchDB gehört zu den dokumentorientierten NoSQL-Datenbanken und überzeugt durch hohe Performance, kinderleichte Replikationsmechanismen und die Möglichkeit, Anwendungen direkt in den Datenbanken vorzuhalten. Mit CouchApps können Entwickler komplette browserbasierte Anwendungen in JavaScript schreiben und ausrollen. Klassische Muster greifen hier nur noch bedingt. Ein Umdenken ist erforderlich, um die Stärken einer unabhängigen CouchApp zu nutzen. Lernen Sie die Möglichkeiten, aber auch die Grenzen von CouchApps kennen, um im Vorfeld die richtige Herangehensweise zu bestimmen.

## 3.10 crypto.is

*Jens Kubieziel, jens@kubieziel.de*

Das Projekt crypto.is versucht, den Ideen der Cypherpunks neues Leben einzuhauchen, und fördert Verschlüsselung und Anonymität. Der Vortrag stellt das Projekt kurz vor und zeigt, was bisher erreicht wurde. Daneben erfahren die Teilnehmer, wie sie am Projekt mitarbeiten können und wem das hilft.

Weitere Informationen: <http://crypto.is/>

## 3.11 Das Anlagen- und Indirekteinleiterkataster der Stadt Bielefeld

*Gerhard Genuit, Stadt Bielefeld, gerd.genuit@arcor.de*

Im Anlagen- und Indirekteinleiterkataster (AUIK) der Stadt Bielefeld werden standortbezogene Daten zu Anlagen zum Umgang mit wassergefährdenden Stoffen und Indirekteinleitern erfasst. Dabei handelt es sich um eine Eigenprogrammierung der Stadt Bielefeld, die mittlerweile von der Intevation GmbH in Osnabrück unterstützt wird. Hier wird auch der Code, der seit April diesen Jahres unter der GNU General Public License steht, gehostet. Die Datenhaltung findet in einer PostgreSQL/PostGIS-Datenbank statt. Über eine Schnittstelle kann QGIS als Geoinformationssystem ausgehend von einem Standort mit einem lokal gespeicherten Projekt geöffnet werden.

Weitere Informationen: <http://wald.intevation.org/projects/auik/>

### 3.12 Datenbanken von MySQL zu PostgreSQL portieren

*Andreas Scherbaum, EMC/Greenplum, ads@pgug.de*

Seit dem Kauf von MySQL durch Oracle sind viele Anwender verunsichert und schauen sich nach Alternativen um. PostgreSQL ist eine dieser Alternativen, sie verzeichnete speziell im letzten Jahr regen Zulauf. Dieser Vortrag zeigt Wege zum Portieren einer Datenbank von MySQL nach PostgreSQL, zusätzlich wird auf häufige Stolpersteine und Probleme eingegangen.

### 3.13 Datensicherheit durch Datensicherung mit bacula

*Karsten Borgwaldt, soIT GmbH, vertrieb@soit.de*

Stetig wachsende Datenmengen und die zeitraubende Prozedur der Sicherung von Daten machen es immer schwerer, ordnungsgemäße und regelmäßige Datensicherungen zu erstellen. Sollte es dann zu einem Datenverlust kommen, so müssen die verloren gegangenen Daten von entsprechenden Spezialisten wiederhergestellt werden, was nicht nur viel Geld, sondern im schlimmsten Fall mehrere Tage Arbeitszeitverlust kostet. Erleben Sie die Umsetzung von Anforderungen an Datensicherheit von mittelständischen Unternehmen mit Open-Source-Mitteln.

Weitere Informationen: <http://www.bacula.org>

### 3.14 Der einfache Umstieg auf Linux mit Kubuntu

*Monika Eggers, kubuntu-de.org*

Wo findet man als Anfänger Hilfe? Was ist überhaupt eine Distribution? Was ist der Unterschied zwischen Ubuntu, Kubuntu und Xubuntu? Welche dieser CDs soll ich nun herunterladen und wie brenne ich sie; oder wie erstelle ich einen Live-USB-Stick? Wie bereite ich meinen Rechner auf die Installation vor? Wie läuft die eigentliche Installation ab? Wie wechsele ich zwischen Netbook-Oberfläche und Desktop-Oberfläche? Was sind die ersten Einrichtungsschritte nach der Installation? Wie installiert man unter Linux Software? Diese und weitere Fragen will der Vortrag beantworten.

Weitere Informationen: <http://www.kubuntu-de.org>

### 3.15 Der Systemaufruf und was danach passiert

*Karl Lockhoff*

Der Anwender startet ein Programm. Das Programm öffnet eine Datei, liest ein paar Daten und schließt anschließend die Datei. Dabei wird zwischen User- und Kernelkontext durch Syscalls gewechselt. Der Vortrag erläutert diesen Prozess genauer.

### 3.16 Developing Linux inside QEMU/KVM Virtual Machines

*Jan Kiszka, Siemens AG, Corporate Technology, jan.kiszka@siemens.com*

Did you ever wonder how to debug a kernel debugger? Or how to get hold of sporadic hard lock-ups in the absence of JTAG-like hardware access? Or if there is a way to shorten turn-around times during driver development? KVM with its user space mate QEMU provides a powerful hypervisor that is also able to make kernel hacking more fun. This talk will introduce to its debugging capabilities. It will present some host/guest setup patterns as well as debugging strategies exploiting SMP, device pass-through, device state visualization, and more.

### 3.17 Die ArchivistaBox kurz erklärt

*Urs Pfister, Archivista GmbH, webmaster@archivista.ch*

Die ArchivistaBox ist eine Linux-basierte Embedded-Box-Lösung mit den Modulen DMS (Dokumentenmanagement), ERP und Virtualisierung. Der Vortrag zeigt auf, wie in maximal zwei Minuten eine ArchivistaBox komplett aufgebaut ist – und vor allem, wie es danach weitergeht. Kurz und gut, der Vortrag stellt die wichtigsten Funktionen der ArchivistaBox in einer halben Stunde vor.

Weitere Informationen: <http://www.archivista.ch/de/pages/support/installation.php>

### 3.18 Die Entwicklung von Linux durch Testen neuer Kernel unterstützen

*Thorsten Leenhuys, Heise Zeitschriften Verlag GmbH & Co. KG, thl@ct.de*

Auch ohne Programmierkenntnisse kann man bei der Entwicklung des Linux-Kernels helfen – insbesondere durch das Testen von noch in Entwicklung befindlichen Kernel-Versionen. Der Vortrag erläutert die wichtigsten Ansätze und Eckpunkte, wie man beim Testen am geschicktesten vorgeht – etwa eigene Kernel kompilieren oder vorkompilierte Kernel aus Paket-Depots für die eigene Distribution einspielen. Ferner wird erläutert, wie man die Kernel-Entwickler über Fehler informiert und mit ihnen die Probleme ausräumt, damit letztlich der Linux-Kernel und die auf ihm aufbauenden Linux-Distributionen besser werden.

### 3.19 Die Helfer der Kommandozeile

*Axel Beckert, ETH Zürich, abe@debian.org*

Der Vortrag ist ein Kommandozeilen-Crashkurs für Einsteiger. Es werden sowohl allgemeingültige Grundlagen als auch konkrete Beispiele für häufig anfallende Arbeiten wie das Kopieren, Verschieben, Umbenennen und Suchen von Dateien gezeigt. Das Mitbringen eines Rechners mit einem Linux oder BSD ist hilfreich, aber nicht notwendig.

Weitere Informationen: <http://noone.org/talks/commandline-helpers/>

## 3.20 Die Macht der Zahlen – Open-Source-Tools für Softwaremetriken

*Gerrit Beine, Gerrit Beine GmbH, mail@gerritbeine.com*

Ob die Qualität von Software mit Hilfe von Metriken beurteilt werden kann, ist immer wieder Grund heftiger Diskussionen. Der Vortrag hat das Ziel, Open-Source-Werkzeuge zur Ermittlung von Metriken vorzustellen. Deren mögliche Aussagen sollen hinterfragt werden. Es werden Werkzeuge für die Sprachen Java, PHP und Perl im Vordergrund stehen. Für die meisten im Vortrag vorgestellten Werkzeuge gibt es aber auch Pendanten in anderen Sprachen.

## 3.21 E-Books mit offenen Standards realisieren

*Benedict Reuschling, The FreeBSD Project*

Elektronische Publikationen lassen sich im freien EPUB-Standard mit Basiskenntnissen von XHTML und CSS ohne viel Mühe erstellen. Veröffentlichungen wie Vereinszeitschriften, Newsletter, Dokumentationen und Abschlussarbeiten sind auch ohne den Kontakt zu einem Verlag möglich. Der Vortrag erläutert den Aufbau von EPUB-Dokumenten anhand eines durchgängigen Beispiels. Neben der grundlegenden Struktur wird dargestellt, wie Text, Bilder und Verweise eingebunden werden. Die Vorstellung geeigneter Open-Source-Werkzeuge rundet den Vortrag ab.

## 3.22 Einfaches Bauen von RPM-Paketen

*Robert Scheck, Fedora Project, robert@fedoraproject.org*

Dieser vorwiegend technische Vortrag zeigt Einsteigern und Fortgeschrittenen, wie sie eigene Programme als RPM-Pakete bereitstellen können. Es werden typische Fehler und Probleme besprochen und geeignete Build-Umgebungen vorgestellt. Der Paketbau wird auf einem Fedora-System gezeigt, funktioniert aber genauso für alle anderen RPM-basierten Distributionen.

## 3.23 Einführung in die 3D-Visualisierung mit Blender

*Erik Schufmann, ROGALL Architekten Ingenieure, info@erikschofmann.de*

In diesem Vortrag soll den Teilnehmern ein Überblick über die Open-Source-3D-Suite Blender vermittelt werden. Nach einer Einleitung zur Geschichte von Blender wird auf die Benutzeroberfläche, die Modellierung von Objekten, die Auswahl von Material und Beleuchtung und auf das eigentliche Rendern der Szenen eingegangen. Alle Themen werden an Beispielen interaktiv erläutert, so dass auf Fragen schnell eingegangen werden kann. Praktische Beispiele runden den Vortrag ab.

Weitere Informationen: <http://www.blender.org>

## 3.24 Fedora Security Lab and the OSSTMM

*Joerg Simon, Fedora Project , [jSimon@fedoraproject.org](mailto:jSimon@fedoraproject.org)*

Der Talk gibt einen Einblick in den momentanen Entwicklungsstand des Fedora Security Lab (FSL) und wie FSL die Methoden und Erkenntnisse aus dem OSSTMM umsetzt und unterstützt und umgekehrt. Wir beleuchten das Information Security Eco System, wie und warum Sicherheitsarbeit durch Standards und Compliance beeinflusst wird und welche Rolle die Security-Solution-Provider dabei spielen. Es werden die neuen Herausforderungen im Bereich Trust Verification betrachtet, die mit den aktuellen Marketing Hypes wie Mobile Security und Cloud kommen und wie man durch den Einsatz von OSSTMM damit umgehen kann.

Weitere Informationen: <http://spins.fedoraproject.org/security/>

## 3.25 Free your slides – Vortragsfolien im Browser anzeigen

*Sirko Kemter, Fedora Project, [gnokii@fedoraproject.org](mailto:gnokii@fedoraproject.org)*

Über die Freiheit von Dateiformaten wird oft geredet, dennoch nutzen nicht wenige Prezi oder ähnliche Plattformen oder stellen ihre Vorträge als PDF bereit. Wie man Vortragsfolien in einem freien Format erstellt, die in jedem Browser darstellbar sind und im Quellformat vorliegen, zeigt diese Präsentation. Scalable Vector Graphic (SVG) ist ein vom W3C betreuter Standard, der von Browsern dargestellt werden kann. Mit ein wenig JavaScript werden daraus animierte Slides, die jenen von Prezi in nichts nachstehen.

## 3.26 Freie Software und Offene Standards – ein starkes Team

*Michael Stehmann, Fellow der FSFE, [info@rechtsanwalt-stehmann.de](mailto:info@rechtsanwalt-stehmann.de)*

Freie Software und Offene Standards werden unter besonderer Berücksichtigung ihrer Vorzüge für den Anwender erläutert. Voraus geht eine Erklärung der Begriffe. Der Vortrag wendet sich an den geschäftlichen und privaten Anwender. Behandelt werden folgende Fragen: Was ist Freie Software? Was sind Offene Standards? Was verbindet Freie Software und Offene Standards? Welche Vorteile bietet Freie Software dem Anwender? Welche Vorteile bieten Offene Standards dem Anwender? Als Fazit ergibt sich, dass Freie Software und Offene Standards eine vorteilhafte Kombination für den Anwender darstellen.

## 3.27 Freifunk – drahtlose Community-Netzwerke

*Klaus Kruse, [kkruse.1987@googlemail.com](mailto:kkruse.1987@googlemail.com)*

Unter dem Label «Freifunk» existiert eine deutschlandweite Bewegung, die vor Ort freie und offene WLAN-Netzwerke aufbaut. Zum Einsatz kommen Standardrouter und angepasste Firmware auf der Basis von OpenWRT. Der Vortrag zeigt die Motivation, wie die Technik funktioniert und wie sich jeder selbst am Aufbau eines Freifunk-Netzwerkes beteiligen kann.

Weitere Informationen: <http://start.freifunk.net/>

### 3.28 FTrace and KernelShark

*Steven Rostedt, Red Hat Inc, rostedt@goodmis.org*

Ftrace is a kernel tracer that lets you analyze what is going on inside the kernel. From seeing almost any function call to more detailed information from static tracepoints. Ftrace's interface can be used with simple echo and cat commands, but this usage can be tedious and the output can be overwhelming. A userspace tool trace-cmd has been created to simplify the interface and create a binary data file, allowing you to parse the data more efficiently. But still the textual output from ftrace makes it difficult to see the bigger picture. This is where KernelShark comes in. KernelShark is a GUI tool that reads the trace-cmd data file and shows the data in a graphical format. This talk will introduce you to trace-cmd as well as KernelShark.

### 3.29 Gib SPAM keine Chance – wirkungsvolle Spamabwehr mit postscreen

*Christoph Wickert, Kolab Systems*

Das Problem unerwünschter Nachrichten ist fast so alt wie die E-Mail selbst. Schon 2002 waren mehr als 50% aller Nachrichten Spam, heute sind es über 90%. Traditionelle Mailfilter stoßen hier an ihre Grenzen, da sie Nachrichten erst aufwendig durchsuchen müssen. Die Postfix-Entwickler haben deshalb mit postscreen einen neuen Ansatz gewählt: Da Spam fast ausschließlich aus Bot-Netzen verschickt wird, versuchen sie, die Spam-Bots zu erkennen. Der Vortrag erklärt, wie man mit postfix, postscreen und anderen Filtern einen wirkungsvollen, mehrstufigen Schutz gegen Spambots und Junkmail einrichtet.

Weitere Informationen: <http://www.postfix.org/>

### 3.30 Globale Wissenstransferprojekte mit KnowledgeWorker

*Lars Fassmann, chemmedia AG, info@chemmedia.de*

Die chemmedia AG bietet mit KnowledgeWorker ein Werkzeug zur Erfassung, multimedialer und interaktiver Aufbereitung, Internationalisierung, Lokalisierung und cross-medialer Auslieferung von Wissensbausteinen auf Basis offener Standards und Technologien (u.a. XML/XSL, Apache Cocoon, Hibernate, jQuery etc.). Eingesetzt wird dieses Werkzeug von internationalen Konzernen bis hin zu mittelständischen Unternehmen und Hochschulen, wenn es darum geht, umfangreiche Wissensinhalte modular aufzubereiten, auf verschiedene Zielgruppen anzupassen und in verschiedenen Formaten bereitzustellen (z.B. HTML, Mobile, PDF, Office-Formate etc.).

### 3.31 Grep everything – geschicktes Suchen in Anwendungsdaten

*Frank Hofmann, Hofmann EDV Berlin, frank.hofmann@efho.de*

*Axel Beckert, abe@debian.org*

Dokumente und Daten erzeugen fällt leicht. Diese später auch wiederzufinden, ist schon etwas schwieriger. Etliche clevere Kommandozeilenwerkzeuge zur Recherche

warten auf ihren Einsatz. Wir zeigen, wie auf der Kommandozeile flink in komprimierten Daten und Archiven gesucht werden kann, ohne diese zuvor auf der Platte auspacken zu müssen. Ebenso durchstöbern wir E-Mails und Mailboxen, PDF-Dokumente und Tabellenblätter. Auf Systemebene gehört das Filtern in Prozesslisten und Netzwerkpaketen dazu.

### **3.32 Hochverfügbarkeit und Virtualisierung – die optimale Kombination beider Konzepte**

*Peter Großöhme, MEGWARE GmbH, peter.grossoehme@megware.com*

Der Vortrag vermittelt die Grundlagen der Hochverfügbarkeit und Virtualisierung, zeigt mögliche Konfigurationen zur Verschmelzung beider Konzepte auf und gibt einen Ausblick auf die neuesten Technologien und Entwicklungen in diesem Bereich.

### **3.33 Integrationstests am Beispiel des Linux-Kernels**

*Frank Becker, fb@alien8.de*

Software muss getestet werden. Neben der reinen Funktionalität spielen auch Parameter wie die Ausführungsgeschwindigkeit oder die verbrauchte Leistung eine wichtige Rolle. Entsprechende Tests und Benchmarks sollen natürlich voll automatisch laufen und ihre Ergebnisse grafisch dargestellt werden. Für diese Aufgabe stellt der Vortrag einige Tools wie Autotest, Codespeed und Jenkins vor, die zu einer Testinfrastruktur verbunden werden können. Als Testobjekt dient exemplarisch der Linux-Kernel. Ziel des Vortrags ist, Anregungen für das automatische Testen einfacher als auch komplexer Systeme zu geben.

Weitere Informationen: <http://autotest.kernel.org/>

### **3.34 Konfiguration und Analyse von Kernel Crashdumps**

*Stefan Seyfried, info@b1-systems.de*

Ist Ihnen jemals ein Linux-Kernel gecrashed, ohne Spuren im Syslog zu hinterlassen? Oder mussten Sie nach einem Kernel-Oops schnellstmöglich das System wieder funktionsfähig bekommen und konnten deswegen die Ursache nicht analysieren? Ein Crashdump wäre in diesen Fällen womöglich hilfreich gewesen. Dieser Vortrag zeigt, wie man Kdump unter Linux konfiguriert und wie man die damit gesammelten dumps untersucht.

Auch wenn Sie kein Kernelhacker sind: die letzte *dmesg*-Ausgabe vom System kann ihnen helfen, die Ursache zu finden (oder jemand anderem die Informationen zu liefern, um Ihnen zu helfen).

### 3.35 Konfigurations- und Infrastrukturmanagement mit RPM und YADT

*Ralph Angenendt, CentOS, Immobilienscout24, ralph+clt@strg-alt-entf.org*

YADT ist ein Deployment- und Managementtool für größere Rechenzentren, das auf RPM aufsetzt. Mit YADT kann eine Systemlandschaft im YAML-Format abgebildet werden. YADT berücksichtigt dabei nicht nur Abhängigkeiten zwischen Paketen, sondern auch die zwischen Services – und das über Rechengrenzen hinweg. Dadurch kann sichergestellt werden, dass bei einem Update einer Plattform die einzelnen Systeme in der richtigen Reihenfolge aktualisiert, konfiguriert und neu gestartet werden. Die Konfigurationen für die einzelnen Systeme werden mit Subversion verwaltet und automatisiert in RPMs überführt.

Weitere Informationen: <https://code.google.com/p/yadt/>

### 3.36 Linux für die kaufmännischen Arbeiten im Unternehmen

*Christian Schuhart, Freier Programmierer, info@ossps.de*

Das Wichtigste im Unternehmen ist das Geldverdienen. Dazu müssen Rechnungen geschrieben werden. Hier erledigt SQL-Ledger seine Arbeit. SQL-Ledger wird seit 1998 von Dieter Simader entwickelt und gehört zu den ältesten Open-Source-Projekten für ERP. Trotzdem hat SL eine überschaubare Dimension in Quellcode und Datenbank. SQL-Ledger war von Anfang an eine echte Webanwendung wie z.B. Joomla oder Typo3. Im Vortrag soll an Beispielen aus der Praxis ein Überblick zur Arbeitsweise und den Möglichkeiten von SQL-Ledger gegeben werden.

Weitere Informationen: <http://www.sql-ledger.com>

### 3.37 LinuXContainer in der Praxis

*Erkan Yanar, erkan.yanar@linsenraum.de*

LinuXContainer (LXC) ist die Virtualisierungslösung des Vanilla Kernels. Trotz vieler Defizite ist sie hervorragend geeignet, sogenannte Applikationscontainer bereitzustellen. Das sind Prozesse mit eigenen Namespaces. Der Vortrag wird einen Einblick in LXC geben und eine Applikationscontainerlösung vorstellen. Anhand der Aufgabe, viele MySQL-Instanzen auf einem Computer zu installieren, wird gezeigt, wie und warum LXC die beste Lösung ist.

### 3.38 Live-Demo: UCS 3.0 und Samba-4-Integration

*Silvia Frank, Univention GmbH, info@univention.de*

In einer Live-Demo wird gezeigt, wie man mit der auf Debian basierenden Linux-Distribution UCS 3.0 ein effektives Domänen- und Identitymanagement in komplexen IT-Umgebungen realisiert. Besonderer Schwerpunkt liegt dabei auf der Vorführung der Integration von Samba 4 sowie der Vorstellung wesentlicher Samba 4-Features. Neu in Samba 4 ist die Unterstützung von Microsofts Active-Directory-Funktionen, so dass zur Benutzer- und Rechteverwaltung oder für das Domänenmanagement kein Windows-Server mehr betrieben werden muss.

### 3.39 LXC oder Jails und Zones

*Westphal Philipp, info@b1-systems.de*

Virtualisierung ist eines der Buzzwords der letzten Jahre. Die meisten eingesetzten Lösungen setzen dabei auf Voll- oder Paravirtualisierung. Dabei entsteht ein riesiger Wasserkopf, z.B. ein XEN-Server, der einen Linux-Server betreibt. Könnten nicht Energie und Hardware effizienter eingesetzt werden, welche anderen Lösungen sind möglich? Und vor allem: Wie mächtig sind diese Process-Isolation-Systeme in Kombination mit modernen Dateisystemen wie HAMMER, btrfs oder zfs?

### 3.40 Mailserver und Spamschutz bei IPv6

*Peer Heinlein, Heinlein Support GmbH, p.heinlein@heinlein-support.de*

IPv6 kommt – okay, seit 10 Jahren. Aber nun hat IPv6 ja wirklich den Einzug in die Rechenzentren genommen, der Produktivbetrieb hat begonnen. Was sich durch IPv6 bei Mailservern und Anti-Spam-Mechanismen ändert, was zu beachten und was total egal ist, ist Thema dieses Vortrages. Und, übrigens: Auch wer nur IPv4 macht, wird an IPv6-Problemen nicht ganz vorbeikommen ...

### 3.41 Menschen reden anders als Maschinen – gepflegter Umgang mit Kritik

*Thomas Rose, Institut für Motivationsanalysen, institut@tachles.cc*

Kritik zu geben ist eine Kunst, aber Kritik anzunehmen ist das tägliche Brot einer jeden Fach- und Führungskraft. Erfahren Sie, wie Sie mit Kritik am besten umgehen. Informieren Sie sich über vier entscheidende Schritte, die Ihnen dabei helfen, mit Kritik an der eigenen Person umzugehen, und mit denen Sie sich möglichst früh in Ihrer Karriere auseinander setzen sollten. Auch wenn Sie im Moment der Liebling aller sind, irgendwann werden Sie sich mit Kritik an Ihrer Person auseinandersetzen müssen.

Weitere Informationen: <http://tachles.cc/2011/01/konstruktiver-umgang-mit-kritik/>

### 3.42 Mit OpenStack zur eigenen Cloud

*Martin Gerhard Loschwitz, hastexo Professional Services GmbH, martin.loschwitz@hastexo.com*

OpenStack ist der vielversprechendste Ansatz, den die Open-Source-Szene in Sachen Cloud Computing zu bieten hat. Dieser Vortrag beginnt mit einer kurzen Einführung in die Prinzipien des Cloud Computings. Im Anschluss erfahren die Zuhörerinnen und Zuhörer alles, was sie über die OpenStack-Cloud-Umgebung wissen sollten. Dazu gehört ein Überblick über die wichtigsten Komponenten – Nova, Glance, Keystone, Swift und Horizon – genauso wie eine ausführliche Beschreibung der aktuell nutzbaren Funktionen. Ein Ausblick auf die Features, die in naher und ferner Zukunft den Umfang von OpenStack noch erweitern werden, rundet die Präsentation ab.

Weitere Informationen: <http://www.openstack.org/>

### 3.43 Mixare.org – eine Augmented Reality Engine

*Patrick Ohnewein, TIS innovation park South Tyrol, [patrick.ohnewein@tis.bz.it](mailto:patrick.ohnewein@tis.bz.it)*

Mixare.org ist eine Augmented Reality Engine für die Betriebssysteme Android und iOS. Das Projekt wurde als Forschungs- und Entwicklungsprojekt von der Internetfirma Peer Internet Solutions aus Südtirol entwickelt und als Freie Software (GPLv3) veröffentlicht.

Weitere Informationen: <http://www.mixare.org>

### 3.44 Monitoring mit Zabbix

*Thorsten Kramm, DV Lösungen Schreiner GmbH*

Der Vortrag gibt zunächst einen Überblick über die Möglichkeiten von Zabbix. Nach der Beschreibung der Installation wird ausführlich auf die Konfiguration eingegangen, die über eine Weboberfläche erfolgt. Neben dem Monitoring von Problemen in großen Netzen wird auch auf die Überwachung weiterer Parameter wie Ressourcenverbrauch eingegangen.

### 3.45 Neue Farbrends

*Kai-Uwe Behrmann, Oyranos, [ku.b@gmx.de](mailto:ku.b@gmx.de)*

An Hand des KDE-Desktops wird in das Farbmanagement mit Oyranos CMS eingeführt. ICC-Farbprofile lassen sich im System von verschiedenen Programmen verwenden. Auf einige kann innerhalb des Vortrages eingegangen werden, und ein paar Fallstricke werden dabei aufgezeigt. Der Farbserver CompICC für die Korrektur aller Farben des Desktops wird vorgeführt.

Weitere Informationen: <http://www.oyranos.org>

### 3.46 NGoC mit Zarafa

*Andreas Rösler, Zarafa Deutschland GmbH, [info@zarafaserver.de](mailto:info@zarafaserver.de)*

Für die Anbindung von MS Outlook, Mobiltelefonen & Co. an Linux-Server ist Zarafa seit Jahren der gesetzte Standard. Doch nun geht Zarafa einen Schritt weiter: Per Mausklick innerhalb einer E-Mail kann diese in einer Videokonferenz diskutiert werden. Weitere nützliche Werkzeuge sind die Verfügbarkeitsanzeige, die Chat-Integration oder die integrierte Dokumentenmanagement-Software. Das alles und noch mehr ist integriert in die Zarafa WebApp – eine Kommunikationsplattform der nächsten Generation. Der Vortrag stellt die Zarafa WebApp vor und demonstriert, wie Kommunikation heute aussehen kann.

Weitere Informationen: <http://www.zarafaserver.de>

### 3.47 Non-free software advertisement presented by your government

*Matthias Kirschner, Free Software Foundation Europe, mk@fsfe.org*

What would you think about a sign on the highway saying «You need a Volkswagen to drive on this road. Contact your Volkswagen dealer for a gratis test drive – Your Government»? When it comes to software that opens PDF files, many public sector organisations do this every day.

With the pdfreaders.org campaign FSFE has turned the spotlight on government organisations who behave in this way, exposing how frequent such advertisements for non-free software are. With the help of activists across Europe, FSFE contacted these organisations and explained them how to improve their websites so that they respect our freedom.

Weitere Informationen: <http://fsfe.org/campaigns/pdfreaders>

### 3.48 Open Data Community für öffentliche Verwaltungen und Unternehmen

*Patrick Ohnewein, TIS innovation park South Tyrol, patrick.ohnewein@tis.bz.it*

Das Interreg-IT-CH-Projekt FreeGIS.net hat das Ziel, eine Referenzimplementierung zur Verwaltung geographischer Daten einzurichten, die zu 100% aus Freien Softwarekomponenten besteht. Weiterhin soll die FreeGIS.net Data License öffentlichen Verwaltungen ermöglichen, ihre GIS-Daten der Allgemeinheit zur Verfügung zu stellen. In FreeGIS.net arbeiten öffentliche Verwaltungen und KMUs gemeinsam an konkreten Lösungen für diese Probleme.

Weitere Informationen: <http://www.freegis.net/>

### 3.49 OpenAFS – ein Überblick für Neulinge

*Lars Schimmer, TU Graz, l.schimmer@cgv.tugraz.at*

OpenAFS ist ein echt-globales Filesystem. Diverse Klienten aus unterschiedlichen Organisationen können weltweit verteilt und synchron auf das selbe Verzeichnis in einem eindeutigen Verzeichnisbaum zugreifen. Dieser Vortrag gibt einen Überblick über die verschiedenen Elemente und Vorzüge dieses Fileservices.

### 3.50 OpenAFS aus Anwender- bzw. Klientensicht

*Andreas Lässer, TU Graz, andreas.laesser@tugraz.at*

OpenAFS ist ein echt-globales Filesystem. Unterschiedliche (Linux,Unix,Windows,Mac) und weltweit verteilte Klienten können damit synchron zusammenarbeiten. Damit dies reibungslos funktioniert, sollte man allerdings ein paar Dinge wissen und beachten. Dieses Grundverständnis wird in diesem Vortrag vermittelt.

### **3.51 OpenAFS-Administration – die andere Seite**

*Markus Köberl, TU Graz, markus.koerberl@tugraz.at*

OpenAFS ist ein echt-globales Filesystem für unterschiedliche Klienten aus unterschiedlichen Organisationen, die weltweit verteilt darauf zugreifen. Dazu ist OpenAFS in «Zellen» organisiert. Typischerweise korreliert eine AFS-Zelle mit einem Kerberos-Realm einer Organisation (z.B. einer Uni). Eine Zelle hat mehrere OpenAFS-Server, die im laufenden Betrieb wechselseitig und stückweise Daten übernehmen oder auch einen Snapshot replizieren können. Wie dies geht (und einiges mehr), zeigt dieser Vortrag.

### **3.52 OpenAFS-Authentisierung – das Kerberos-Basisprotokoll zum Anfassen**

*Mathias Feiler, KIM der Universität Hohenheim, feiler@uni-hohenheim.de*

OpenAFS ist ein echt-globales Filesystem für unterschiedliche Klienten, die weltweit verteilt darauf zugreifen. Dies geht natürlich nicht ohne Authentisierung. Dazu verwendet OpenAFS seit mehr als 20 Jahren Kerberos. Es ist daher sinnvoll, dieses Protokoll in seinen Grundzügen zu verstehen. Zur Veranschaulichung wird am Ende des Vortrags das Basis-Kerberos-Protokoll als Theaterstück «Der Ur-Kerberos» von einigen engagierten Zuhörern durchgespielt werden.

### **3.53 Penetration Testing mit Metasploit**

*Stefan Schumacher, Institut für Sicherheitsforschung, Info@Sicherheitsforschung-Magdeburg.de*

Metasploit ist ein Open-Source-Framework, das Informationen über Sicherheitslücken sammelt. Damit eignet es sich, um Einbruchsversuche gegen Rechner zu fahren (sog. Penetration Testing) und diese so auf Schwachstellen zu überprüfen. Weiterhin kann man Metasploit in der Sicherheitsforschung und zur Entwicklung von Signaturen für Intrusion-Detection-Systeme verwenden.

### **3.54 Programmierung für Smartphones – die Lizenzen für Android und iOS erläutert**

*Christian Klostermann, Rechtsanwaltskanzlei Dr. Klostermann, kanzlei@drklostermann.de*

Smartphones boomen, und die Programmierung von Apps ist dank vorhandener Entwicklungswerkzeuge schnell und effizient möglich. Doch welche rechtlichen Vorschriften sind zu beachten, wenn man den Android App Market oder iOS App Store nutzen möchte? Unter welcher Lizenz steht die Nutzung der Software? Welches Geschäftsmodell für die eigene Software kann man wählen? Der Vortrag erläutert die rechtlichen Bedingungen und richtet sich vornehmlich an Firmen und Entwickler von Apps für Smartphones oder solche, die es werden wollen. Vorkenntnisse sind nicht erforderlich.

### 3.55 Python – Open-Source-Werkzeug für Wissenschaftler und Ingenieure

*Mike Müller, Python Academy, Leipzig Python User Group, mmueller@python-academy.de*

Wissenschaftler und Ingenieure lösen viele Probleme mit dem Computer und müssen dafür häufig programmieren. Python hat sich in den letzten Jahren als wichtige Sprache für derartige Anwendungen etabliert. Neben den Vorzügen der Sprache wie leichte Erlernbarkeit und Ausdrucksstärke spielen vor allem die zahlreichen, in hoher Qualität vorliegenden Open-Source-Bibliotheken eine große Rolle für die Beliebtheit von Python.

Weitere Informationen: <http://www.python.org>

### 3.56 QEMU's recompilation engine

*Alexander Graf, agraf@suse.de*

Have you ever wondered how emulators work? How they translate code from one architecture to another? QEMU can even run on any modern host architecture, without reinventing the wheel every time. But how does this work? This talk will give you an overview of TCG – the small compiler used by QEMU – and the general architecture of instruction emulation in QEMU. It will show you how guest code is analyzed and how host code is emitted from that. It will also show you a few performance optimization tricks it does.

Weitere Informationen: <http://www.qemu.org>

### 3.57 Qualitätsanalyse und Teammanagement in Open-Source-Projekten

*Andreas Tille, Debian GNU, tille@debian.org*

Die Bildung eines schlagkräftigen Teams aus den Reihen einer Nutzergemeinschaft ist eine essentielle Voraussetzung für ein erfolgreiches Softwareprojekt. Daher wurde im GSoCode Projekt «Debian Teams Activity Metrics» analysiert, wie gut Gruppen innerhalb von Debians Unterprojekten zusammenarbeiten. Das vorzustellende Analyseprinzip und der zugrundeliegende Code ist auf andere Projekte übertragbar. Neben der Teamanalyse geht der Vortrag auf typische Probleme ein, die verhindern, dass ein Open-Source-Projekt eine Community anzieht, und somit eine erfolgreiche Teambildung ausschließen.

Weitere Informationen: <https://www.theopensourceway.org/wiki/>

### 3.58 Rechnungseingangsverarbeitung mit Alfresco

*Bernt Penderak, soIT GmbH, vertrieb@soit.de*

Alfresco bietet Ihnen echtes Open Source Enterprise Content Management (ECM): Document Management, Collaboration, Records Management, Knowledge Management, Web Content Management und Imaging. Wir zeigen Ihnen einen Rechnungseingangs-Workflow aus dem betrieblichen Alltag – mit Freier Software.

Weitere Informationen: <http://www.soit.de/produkte-und-loesungen/>

### 3.59 SAN Storage on Linux

*Hannes Reinecke, SuSE Labs, hare@suse.de*

This presentation gives an introduction into SAN (Storage Attached Network) and the implementation on Linux. In particular it focuses on FibreChannel (FC) and its most recent implementation, FibreChannel over Ethernet (FCoE).

Storage Attached Networks (SAN) have been common in enterprise environments for a long time, however, the hardware demands made it virtually unknown outside dedicated serverrooms.

With 10GigE and its FCoE capabilities it becomes possible to use FC without dedicated equipment. So it's about time to familiarize ourselves with FC and its implementation on Linux.

### 3.60 SocketCAN – CAN-Treiberschnittstelle unter Linux

*Daniel Krüger, SYS TEC electronic GmbH, info@systec-electronic.com*

SocketCAN ist eine Programmier- und Treiberschnittstelle für Controller-Area-Network-Geräte. Diese Schnittstelle basiert auf dem BSD Socket API. CAN wird vor allem im industriellen und automobilen Bereich eingesetzt. Im Vortrag wird in die Grundlagen von CAN eingeführt. Weiterhin wird gezeigt, wie SocketCAN in eigenen Applikationen unter Linux angewendet wird. Natürlich gibt es auch eine Live-Demo.

Weitere Informationen: <https://gitorious.org/linux-can>

### 3.61 Spacewalk

*Christian Berendt, B1 Systems GmbH, info@b1-systems.de*

Mit Spacewalk lassen sich Server und Clients warten, Software verteilen und neue Systeme automatisch installieren (Provisioning). Das Einsatzgebiet ist dank Weiterentwicklungen nicht auf Red-Hat-basierte Distributionen wie RHEL, CentOS oder Fedora beschränkt, sondern wurde auf Distributionen wie OpenSUSE und SLES erweitert. Die B1 Systems GmbH gibt in diesem Vortrag einen ausführlichen Einblick in die Fähigkeiten von Spacewalk, zeigt die Vorteile und wie Sie mit dieser Open-Source-Lösung Softwarebudgets und Ressourcen effizient einsetzen.

### 3.62 Speichereffiziente Datensicherung mit Dirvish

*Robert Sander, Epigenomics AG, clt2012@gurubert.de*

Dirvish ermöglicht mit Hilfe von rsync die speichereffiziente Datensicherung von Filesystem-Snapshots. Jeder einzelne Snapshot enthält dabei den kompletten Verzeichnisbaum, trotzdem kann bei geringer Änderungsrate eine lange Historie behalten werden. Vorgestellt werden die Features von Dirvish im Zusammenspiel mit LVM-Snapshots und im Vergleich zu ähnlichen Tools wie rdiff-backup oder rsnapshot.

### 3.63 Stop-Motion-Trickfilme mit Linux

*Ralf Lange, ralf.lange@qstopmotion.org*

Wer schon immer mal seinen eigenen Stop-Motion-Trickfilm drehen wollte, aber vor den technischen Herausforderungen zurückschreckte, erhält in diesem Vortrag einen Überblick zur benötigten Hard- und Software und viele Tips für deren Anwendung. Die Aufnahme der Bilder und das Erstellen eines Videos werden kurz vorgeführt.

Weitere Informationen: <http://www.qstopmotion.org>

### 3.64 Supercomputing and Open Source after the K-Computer

*Christian Kuelker, ETH Lab, c.kuelker@ethlab.com*

The focus of supercomputing after the K-Computer reached number 1 of the TOP 500 is still on the question how it is possible to use all this performance. The Japanese have some answers to the question. The lecture will also give some insights about the current developments and streams regarding the SC11 in Seattle.

### 3.65 TaskJuggler 3.0 – Projektmanagement für Linux-Anwender

*Chris Schlaeger*

Das TaskJuggler-Projekt wurde vor 10 Jahren als Open-Source-Projekt gestartet, als SuSE ein leistungsfähiges Projektmanagementprogramm zur Planung und Überwachung der Distributionsentwicklung brauchte. Heute wird es von vielen Firmen und Privatnutzern weltweit eingesetzt. TaskJuggler ist eines der wenigen Projektmanagementprogramme, die vollständig auf die Linux-Umgebung zugeschnitten sind. Im Oktober 2011 wurde die nächste Generation der Software, TaskJuggler 3.0, veröffentlicht. Dieser Vortrag bietet einen Einstieg in das Arbeiten mit TaskJuggler und einen Überblick über die vielen Neuerungen der aktuellen Generation.

Weitere Informationen: <http://www.taskjuggler.org>

### 3.66 The 3.0-rt Kernel

*John Kacur, Red Hat, jkacur@redhat.com*

For a long time the Linux Real-Time developers were stuck maintaining their patchset on the 2.6.33 kernel. Then, thanks largely to a handful of folks and especially to Thomas Gleixner, the Real-Time patch set was ported to 3.0. This talk gives a brief introduction to the Real-Time kernel and talks about some of the changes involved porting the Real-Time patchset to 3.0

### 3.67 Tine 2.0 – Open-Source-Groupware und CRM

*Lars Kneschke, Metaways Infosystems GmbH, l.kneschke@metaways.de*

Tine 2.0 ist eine webbasierte Open-Source-Groupware-Lösung mit dem Fokus auf Anwenderfreundlichkeit und Stabilität. Tine 2.0 ermöglicht es Anwendern, Termine,

Kontakte und Aufgaben gemeinsam über eine moderne Weboberfläche zu verwalten. Der Zugriff auf die Daten vom Mobiltelefon ist ebenfalls ohne Probleme möglich. Darüber hinaus werden auch die weiteren Module wie Zeiterfassung, CRM und die grundlegenden Bedienkonzepte von Tine 2.0 erklärt.

Der Vortrag richtet sich primär an Anwender, die eine Lösung für das gemeinsame Verwalten von Daten suchen.

Weitere Informationen: <http://www.tine20.org>

### 3.68 Typo3 für Entwickler

*Roman Geber, B1 Systems GmbH, [info@b1-systems.de](mailto:info@b1-systems.de)*

Bekannt wurde Typo3 als Enterprise Content Management System, doch wird diese Bezeichnung dem Potential von Typo3 nicht gerecht. Für Entwickler stellt Typo3 eine vielseitige Plattform zur Entwicklung moderner Online-Anwendungen dar. Zum Funktionsumfang gehören unter anderem Template Engines, ein administratives Backend, Benutzerverwaltung und ein Extension Manager sowie tausende frei verfügbare Erweiterungen. Wege zum Ziel gibt es meist viele. Klare Projektstrukturen, penible Trennung von Inhalt und Präsentation, Verwendung eingebauter APIs und Einhaltung einiger «Best Practices» führen zu einem saubereren Ergebnis in kürzerer Zeit.

### 3.69 Ubuntu im sicheren privaten, virtuellen Netz

*Richard Albrecht, [richard@leofield.de](mailto:richard@leofield.de)*

Warum sollen wir GNU/Linux/Ubuntu einsetzen? Warum ist PC-Sicherheit so wichtig? Private Daten und Vorgänge erledigen wir heute mit dem PC. Daten tauschen wir mit Freunden und in der Familie aus, oft über sehr unsichere Systeme. Mit Ubuntu ist es leicht möglich, die Sicherheit eines PCs zu erreichen und ein einfaches und sicheres privates Netz aufzubauen. Es wird gezeigt, wie dies auf SSH-Basis geschieht und wie auch Windows-Systeme eingebunden werden können. Ziel ist es, Hilfe zur Selbsthilfe zu geben.

Weitere Informationen: <http://lug-ottobrunn.de/wiki/Kategorie:Linuxeinsteiger>

### 3.70 Ursprünge der Versionsverwaltung und Revival von SCCS

*Jörg Schilling*

SCCS ist die älteste Versionsverwaltung (eingeführt 1972). Seit Dezember 2006 ist SCCS Open Source und wird weiterentwickelt. Zur Zeit ist SCCS auf dem Wege zu einem netzwerkfähigen, verteilten Versionsverwaltungssystem. Der Vortrag erklärt die Ziele und den Stand der aktuellen Entwicklung sowie die Vorteile des SCCS-Dateiformats bei der Verwaltung von großen, langlebigen Projekten. Es wird auch auf die Geschichte der Entwicklung von SCCS eingegangen und mit der Entwicklung von projektorientierten Systemen wie NSE, TeamWare, BitKeeper, Git, Mercurial in Zusammenhang gebracht.

Weitere Informationen: <http://sccs.berlios.de>

### 3.71 Virtual System Cluster: Freie Wahl mit Open Source

*Ralph Dehner, B1 Systems GmbH, info@b1-systems.de*

B1 Systems stellt eine Pacemaker-basierte Virtual-System-Cluster-Lösung vor, die den Nutzer nicht langfristig an eine bestimmte Technologie oder einen Hersteller bindet. Der Vortrag gibt einen Überblick über die verschiedenen Möglichkeiten, einen Virtual System Cluster mit Open-Source-Mitteln aufzubauen, um Linux oder Windows in virtuellen Maschinen hochverfügbar zu betreiben. Vorgestellt werden VSC auf der Basis von Xen sowie KVM.

### 3.72 Virtualisierung @ Google

*Alexander Schreiber, Google Switzerland GmbH, als@google.com*

Google verwendet Virtualisierung für verschiedene Zwecke: Serverkonsolidierung, virtuelle Workstations und als Werkzeug zum Maschinenmanagement. Der Vortrag wird einen Überblick über die verwendeten Technologien, die Organisation der Systeme auf Maschinen-, Cluster- und Flottenebene und interne Tools bieten. Zudem werden Workflows für die Zuweisung und Rückgabe virtueller Maschinen, den Umgang mit ausgefallenen Maschinen sowie die Handhabung neuer oder reparierter Maschinen dargestellt. Ein Ausblick auf die künftige Weiterentwicklung der verwendeten Software schließt den Vortrag ab.

Weitere Informationen: <http://code.google.com/p/ganeti/>

### 3.73 Vom Piep zum Boot – BIOS und Co.

*André Przywara, AMD Dresden*

Aufgabe der Firmware – beim PC oft BIOS genannt – ist das Initialisieren des Rechners. Der Vortrag wird den Vorgang etwas genauer erläutern und dabei die einzelnen Schritte und Software-Bestandteile benennen. Dabei wird auch UEFI eine Rolle spielen. Nicht fehlen wird auch der Bootloader Grub, der das Laden des eigentlichen Kernels erledigt.

### 3.74 Web Framework Django

*Thomas Güttler, TBZ-PARIV, guettli.ct12@thomas-guettler.de*

Der Vortrag gibt einen Überblick über das freie Web Framework Django, das die Entwicklung von Web-Applikationen mit der Programmiersprache Python ermöglicht. Es bietet im Kern einen objektrelationalen Mapper und eine HTML Template Engine. Django unterstützt Unicode und beinhaltet ein Admin-Interface für die Datenbank. Im Rahmen des Vortrags soll Django aus der Vogelperspektive vorgestellt werden.

Weitere Informationen: <http://www.thomas-guettler.de/vortraege/django/>

### **3.75 Wollmux trifft auf SAP & Co – dynamische Reports mit dem freien Vorlagensystem**

*Frank Siebert, Trinuts GmbH, fsiebert@trinuts.de*

Seit Mitte 2008 steht das flexible Vorlagensystem Wollmux auf der OSOR-Plattform der EU zum Download bereit. Vorlagen und Formulare lassen sich damit auf einfache Weise zentral pflegen und für alle Mitarbeiter bereitstellen. Anhand der Formular- und Reportgenerierung in SAP wird gezeigt, wie sich Wollmux zur zentralen und komfortablen Ausgabeinstanz für Dokumente aus behörden- und unternehmensspezifischen Anwendungen erweitern lässt. So lassen sich einfach die Pflege der ODF-Formulare und ein einheitliches CI kombinieren und gleichzeitig die komfortablen Funktionen von Wollmux nutzen.

Weitere Informationen: <http://www.trinuts.de>

### **3.76 Zentrales Konfigurationsmanagement mit Puppet**

*Martin Alfke, Wizards of FOSS*

Jeder Administrator einer großen Serverlandschaft steht regelmäßig vor der Aufgabe, eine oder mehrere Änderungen auf viele Server zu verteilen. Hierbei ist ein zentrales Konfigurationsmanagement hilfreich, das sicherstellt, dass die gewünschten Änderungen auch auf neue Systeme übernommen werden. Dieser Vortrag gibt einen kurzen Überblick über vorhandene Werkzeuge und zeigt anschließend die Funktionsweise und detaillierte Beispiele von Puppet. Zum Abschluss erfolgt eine grobe Übersicht über den Einsatz von Puppet bei der Continental Automotive GmbH.

Weitere Informationen: <http://puppetlabs.com/>

## 4 Zusammenfassungen der weiteren Workshops

### 4.1 Arduino – Ausbruch aus dem Compile-Upload-Debug-Kreislauf

*Carsten Strotmann, Forth Gesellschaft e.V.*

Der Arduino ist ein beliebtes Open-Source-Hardwareboard auf der Basis des Atmel AVR-Mikrocontrollers. Hardwareentwicklung auf dem Arduino ist häufig an einen Compile-Upload-Debug-Zyklus gebunden. Die Entwicklung findet auf einem PC und nicht auf der Zielhardware statt. Dieser Workshop zeigt, wie Programmierung mit amForth direkt auf dem Arduino geschehen kann, ohne dass ein Cross-Compile-Prozess notwendig ist. Die Hardware kann interaktiv auf der Kommandozeile erforscht und programmiert werden.

Weitere Informationen: <http://amforth.sourceforge.net/>

### 4.2 Einführung in Blender

*Andreas Löscher, TU Chemnitz*

Blender ist eine 3D Content Creation Suite, mit der man von einfachen Modellen bis zu ganzen Filmen alles erstellen kann, was das Herz begehrt. Der Workshop bietet die Gelegenheit, ohne Vorwissen das Erstellen einfacher Szenen zu erlernen. Dabei wird neben einer grundlegenden Einführung in die Bedienung und Modellierung eine komplette Szene kreiert und gerendert.

Weitere Informationen: <http://www.blender.org/>

### 4.3 Einstieg in die Betriebssystementwicklung – Grundlagen für das eigene OS

*Tobias Stumpf, ESAS Student Group Chemnitz, [cs@tobias-stumpf.de](mailto:cs@tobias-stumpf.de)*

*Mathias Keller, [matthias.keller@s2007.tu-chemnitz.de](mailto:matthias.keller@s2007.tu-chemnitz.de)*

Wer immer schon mal ein eigenes Betriebssystem entwickeln wollte, ist hier genau richtig. Aber auch Entwickler von hardwarenaher Software, die gern auf die ein oder andere Funktionalität eines Betriebssystems zurückgreifen wollen, werden nützliche Grundlagen finden. Ziel ist es, einen kleinen Kernel für die x86-Architektur zu entwickeln. Zu Beginn lernen die Teilnehmer, wie sie die Hardware initialisieren und erste Gerätetreiber für ihr eigenes Betriebssystem schreiben. Im Laufe der Entwicklung werden immer mehr Funktionen bereitgestellt, die die spätere Programmierung von Anwendungen erleichtern.

Weitere Informationen: <http://www.tobias-stumpf.de/OSWorkshop/index.html>

## 4.4 FreeBSD-Installation und -Konfiguration

*Benedict Reuschling, The FreeBSD Project*

In diesem Workshop geht es darum, FreeBSD kennen- und installieren zu lernen. Wir werden das aktuelle FreeBSD-Release mit dem neuen Installer auf den Notebooks der Teilnehmer (als virtuelle Maschinen) installieren und dabei wichtige Details zum System erklären. Nach der Installation werden die wichtigsten Schritte zur Konfiguration erläutert, und es wird gezeigt, wie man weitere Software aus den Ports installiert.

Weitere Informationen: <http://www.FreeBSD.org>

## 4.5 Inkscape – Eiszeit

*Sirko Kemter, Fedora Project, gnokii@fedoraproject.org*

Inkscape ist ein freier Vektorgrafikeditor. Dass man mit ihm mehr als ein paar schöne Kreise zeichnen kann, zeigt dieser Workshop. Inkscape verwendet den SVG-Standard des W3C, damit sind die Grafiken in nahezu jedem Browser darstellbar. Der nächste Sommer kommt bestimmt, und wenn es draußen heiß ist, dann schmeckt ein selbstgezeichnetes Eis ganz sicher.

Weitere Informationen: <http://inkscape.org>

## 4.6 KMUX – Installation der IT-Landschaft für ein KMU in 2 Stunden

*Julian Thomé, KMUX-Projekt*

In diesem Workshop möchte ich zeigen, wie man mit einer Handvoll Kommandos die komplette IT-Infrastruktur und alle Anwendungen für ein kleines Unternehmen oder eine gemeinnützige Einrichtung installiert und nutzbar macht. Innerhalb des Workshops erläutere ich das technische KMUX-Konzept, und während dieser Zeit werden wir die Installation eines «ganzen Unternehmens in einer Box» auch durchführen.

## 4.7 Offenes Storage Management mit openATTIC

*David Breitung, openATTIC / it-novum, david@open-attic.org*

openAttic ist ein neues Projekt für Storage Management. Es eignet sich für den Betrieb eines hochverfügbaren Rechenzentrums. openAttic ist ein hochflexibles, zentrales Framework, das verschiedene Storage Tools unter einer webbasierten Oberfläche vereint. Im Workshop wird die Weiterentwicklung von openAttic besprochen und diskutiert.

## 4.8 OpenAFS – eine eigene AFS-Zelle aufsetzen

*Lars Schimmer, TU Graz, l.schimmer@cgv.tugraz.at*

OpenAFS ist ein echt-globales Filesystem, das unterschiedliche Klienten unterstützt. Es ist in Zellen organisiert. Das erste Aufsetzen einer AFS-Zelle ist allerdings nicht ganz trivial. Darum bieten wir als Ergänzung zur Vortragsreihe (OpenAFS) diesen

Workshop an. Voraussetzung zur sinnvollen Teilnahme sind Root-Zugang auf mindestens ein gängiges aktuelles Linux in einer virtuellen Maschine (vorzugsweise Ubuntu/Debian), eine freie (virtuelle) Partition mit mindestens 5 GB Größe, Netzverbindung und fundierte Kenntnisse in der Unix-Administration.

## 4.9 OpenFOAM: Numerik mit freier Software am Beispiel

*Steffen Weise*

OpenFOAM ist ein in C++ geschriebenes freies Softwarepaket zur Lösung von numerischen Problemen, bevorzugt aus dem Bereich der Strömungsmechanik. Nach der Erläuterung des Aufbaus der Software wird gezeigt, wie man Gitter für numerische Berechnungen erzeugt, Randbedingungen und Parameter für Simulationen festlegt und deren Ergebnisse auswertet und mit Paraview visualisiert.

Weitere Informationen: <http://www.openfoam.com/>

## 4.10 OpenStreetMap

*Andreas Tille, Debian GNU, [tille@debian.org](mailto:tille@debian.org)*

*Thomas Bellmann, [osm@malenki.ch](mailto:osm@malenki.ch)*

*André Riedel*

Der Workshop soll die Teilnehmer befähigen, aktiv Daten in die OpenStreetMap-Datenbank einzupflegen. Dabei werden sowohl einfache Merkmale als auch Routenrelationen behandelt. Weiterhin werden Tools vorgestellt, um im Datenbestand von OpenStreetMap nützliche Informationen für das Mappen abzufragen.

Weitere Informationen: [http://malenki.ch/clt\\_2012/ws](http://malenki.ch/clt_2012/ws)

## 4.11 Python für Einsteiger

*Stefan Schwarzer, [SSchwarzer.com](http://SSchwarzer.com), [sschwarzer@sschwarzer.com](mailto:sschwarzer@sschwarzer.com)*

*Thomas Güttler, TBZ-PARIV Archiv-, Workflow- und Posteingangssysteme, [guettli.clt12@thomas-guettler.de](mailto:guettli.clt12@thomas-guettler.de)*

Der Workshop bietet eine Einführung in die Programmiersprache Python. Die Sprache ermöglicht kompakte, gut lesbare Programme für Systemadministration, Web, Wissenschaft und viele andere Gebiete. Python ist auch eine ausgezeichnete Sprache zum Verbinden verschiedener Systeme. In Python lässt sich prozedural und objekt-orientiert programmieren.

Weitere Informationen: <http://www.python.org>

## 4.12 Seeing inside the Linux Kernel with ftrace

*Steven Rostedt, Red Hat Inc, [rostedt@goodmis.org](mailto:rostedt@goodmis.org)*

This workshop will show people how to use ftrace to see what the Linux kernel is doing. Ftrace is an internal kernel tracer that allows you to see what functions the

kernel is executing. Users can see functions as well as trace points, such as, tasks scheduling, timers, interrupts, and much more. Ftrace is enabled in most distribution kernels, so attendees do not need to install a new kernel to participate. Attendees will also learn how to use trace-cmd and kernelshark that are tools to interact with ftrace. Weitere Informationen: <http://people.redhat.com/srostedt>

#### **4.13 Starbasic – eine Einführung**

*Michael Stehmann, Fellow der FSFE, [info@rechtsanwalt-stehmann.de](mailto:info@rechtsanwalt-stehmann.de)*

Starbasic ist die Makro-Programmiersprache von OpenOffice.org und LibreOffice. Makros können den Büroalltag erleichtern. Der Workshop will eine Einführung in die Makroprogrammierung geben und ihre Möglichkeiten und die Ressourcen, auf die ein Programmierwilliger zurückgreifen kann, aufzeigen. Praktische Beispiele sollen diese Möglichkeiten veranschaulichen.

Weitere Informationen: <http://de.openoffice.org>

#### **4.14 Vim-Führerschein: Grundlegendes Editieren mit Vim**

*Jana Wisniowska, [janapirat@gmx.de](mailto:janapirat@gmx.de)*

Wer sich tiefergehender mit dem Linux-System befassen möchte, wird um den Texteditor Vim nicht herumkommen. In diesem Workshop werden grundlegende Konzepte erklärt und einfache Editieraufgaben spielerisch geübt. Es wird auf typische Anfängerschwierigkeiten eingegangen. Der Workshop widmet sich denjenigen, die noch nie mit dem Vim-Editor gearbeitet haben oder sich noch unsicher fühlen. Lernziel: einfache Editierarbeiten unfallfrei erledigen.

## Personen

- Albrecht, Richard, 164  
Alfke, Martin, 166  
Angenendt, Ralph, 156  
Aßmann, Uwe, 45
- Bäcker, Renée, 149  
Barth, Rico, 148  
Becker, Frank, 155  
Beckert, Axel, 148, 151, 154  
Behrmann, Kai-Uwe, 158  
Beine, Gerrit, 152  
Bellmann, Thomas , 169  
Berendt, Christian, 162  
Berger, Uwe, 35  
Borgwaldt, Karsten , 150  
Breitung, David, 168
- Dehner, Ralph, 165
- Eggers, Monika, 150
- Fassmann, Lars, 154  
Feiler, Mathias , 160  
Frank, Silvia, 156
- Gachet, Daniel, 77  
Geber, Roman, 149, 164  
Genuit, Gerhard, 149  
Graf, Alexander, 161  
Großöhme, Peter , 155  
Güttler, Thomas, 165, 169  
Götz, Sebastian, 45
- Heinlein, Peer , 157  
Hofmann, Frank, 154
- Jamous, Naoum, 137
- Kacur, John, 163  
Kastrup, David, 61  
Keller, Mathias, 167
- Kemter, Sirko, 153, 168  
Kirschner, Matthias, 159  
Kiszka, Jan, 151  
Klostermann , Christian, 160  
Kneschke, Lars, 163  
Knopper, Klaus, 117  
Köberl, Markus, 160  
Kölbel, Cornelius, 87  
König, Harald, 99  
Kramer, Frederik, 137  
Kramm, Thorsten, 158  
Krüger, Daniel, 162  
Kruse, Klaus, 153  
Kubieziel, Jens, 21, 149  
Kuelker, Christian, 163
- Lässer, Andreas, 159  
Lang, Jens, 69  
Lange, Ralf, 163  
Leemhuis, Thorsten, 147, 151  
Leuthäuser, Max, 45  
Lockhoff, Karl, 150  
Löscher, Andreas, 167  
Loschwitz, Martin Gerhard, 157  
Luithardt, Wolfram, 77
- Meier, Wilhelm, 13, 131  
Müller, Mike, 161
- Nasrallah, Olivier, 77  
Neitzel, Martin, 148
- Ohnewein, Patrick, 158, 159
- Penderak, Bernt, 161  
Pfister, Urs, 151  
Philipp, Westphal, 157  
Piechnick, Christian, 45  
Przywara, André, 165
- Reimann, Jan, 45

Reinecke, Hannes, 162  
Reuschling, Benedict, 152, 168  
Richly, Sebastian, 45  
Riedel, André, 169  
Rösler, Andreas, 158  
Rose, Thomas, 157  
Rostedt, Steven, 154, 169

Sander, Robert, 162  
Scheck, Robert, 152  
Scherbaum, Andreas, 150  
Schilling, Jörg, 164  
Schimmer, Lars, 159, 168  
Schlaeger, Chris, 163  
Schöner, Axel, 13  
Schreiber, Alexander, 165  
Schroeter, Julia, 45  
Schütz, Georg, 53  
Schufmann, Erik, 152  
Schuhart, Christian, 156  
Schumacher, Stefan , 160  
Schwarzer, Stefan, 169  
Seyfried, Stefan, 155  
Siebert, Frank, 166  
Simon, Joerg, 153  
Stehmann, Michael, 153, 170  
Strotmann, Carsten, 167  
Stumpf, Tobias, 167

Thomé, Julian, 168  
Tille, Andreas, 161, 169

Vorwerk, Matthias, 109

Wachtler, Axel, 109  
Weisbecker, Frederic, 147  
Weise, Steffen, 169  
Wickert, Christoph, 154  
Wilke, Claas, 45  
Winde, Thomas, 147  
Wirtz, Markus, 148  
Wisniowska, Jana, 170  
Wunsch, Jörg, 109

Yanar, Erkan, 156