

Freie Software Kultur und Technik

Peter Husemann

Marcel Martin

Ingo Ruhnke

Henner Sudek

30. März 2002

Inhaltsverzeichnis

1	Motivation	3
2	Grundgedanke der Freien Software	3
3	Geschichte	4
3.1	Das GNU-Projekt	4
3.2	Der Hurd-Kernel	4
3.3	Linux	5
3.4	BSD	6
3.5	Apache	6
4	Lizenzen für Freie Software	7
4.1	GPL – GNU General Public License	9
4.2	LGPL – Lesser General Public License	9
4.3	BSD – Berkeley Software Distribution	10
4.4	Apache	10
4.5	QPL – Q Public License	10
5	Funktioniert Freie Software überall?	11
5.1	Office, E-Mail und Co.	11
5.2	Server-Software	12
5.3	High-End-Software	12
5.4	Spiele	12
6	Freie Software an Schulen	14
7	Gefahren für freie Software	15
7.1	Patente	15
7.1.1	Das europäische Patentrecht	16
7.1.2	Patentpraxis in den U.S.A.	17
7.1.3	Überlastete Patentämter	17
7.2	Proprietäre Standards	18
8	Businessmodelle	19
8.1	Distributoren	19
8.2	Support	20
8.3	Kleine Unternehmen	20
8.4	Subscription-Modell	21
8.5	Verzögerte Releases	22
8.6	Fazit	23

<i>INHALTSVERZEICHNIS</i>	2
8.7 Vor- und Nachteile Freier Software	23
8.7.1 Innovationsarmut	23
8.7.2 Usability	24
8.8 Maintenance	24
8.9 Neue Technik	25
9 Open Source und die Gesellschaft	25
9.1 Wissenschaft	27
10 Zukunft	27
10.1 Informationsgesellschaft	28

1 Motivation

Die meisten der Autoren dieses Dokuments haben bereits selbst an der Entwicklung und Verbesserung von sogenannter freier Software teilgenommen oder sogar eigene Projekte ins Leben gerufen. Als aktive Teilnehmer der Open-Source-Bewegung geht es uns nicht nur darum, dieser Gemeinschaft durch unsere Programmierfertigkeiten etwas zurückzugeben, sondern insbesondere auch darum, die Strukturen, Prinzipien, ungeschriebenen Gesetze und ihre möglichen Implikationen für die Nicht-Programmiererwelt zu verstehen.

Wir werden in den ersten Abschnitten die wichtigsten Elemente dieser Community erläutern, die zum Verständnis der nachfolgenden Kapitel nötig sind. Dort werden dann eher wirtschaftliche, rechtliche und soziale Aspekte behandelt. Zum Schluss erlauben wir uns einen Ausblick, wie es um die Zukunft der freien Software bestellt ist. (mm)

2 Grundgedanke der Freien Software

Zu Beginn der Computerentwicklung waren die Quelltexte zu allen Programmen offen. Jeder konnte die Software seinen eigenen Bedürfnissen anpassen, kopieren und sowohl veränderte als auch unveränderte Versionen an andere weitergeben. Dies war ebenso selbstverständlich wie es heute für uns ganz normal ist täglich mit Software zu arbeiten, deren Quelltexte nicht einzusehen sind. Mit der zunehmenden Verbreitung von Computersystemen und der damit steigenden kommerziellen Bedeutung von Software änderte sich dies jedoch sehr deutlich. Firmen gingen immer mehr dazu über den Quelltext ihrer Software als ein Geheimnis zu betrachten, das niemand außer ihnen erfahren durfte.

Die Idee hinter der Freien Software ist es, dem Benutzer die Rechte wiederzugeben, die die meisten Softwarehersteller ihnen genommen haben: Das freie Verteilen und Verändern der Software. Dieser Gedanke hat bis heute auf der ganzen Welt eine wachsende Zahl von Anhänger gefunden. Dieser Gemeinschaft von Programmierern ist es zu verdanken, dass heute eine ganze Reihe höchst erfolgreicher Freier-Software-Projekte existieren, von denen wir die wichtigsten im weiteren Verlauf der nächsten Abschnitte vorstellen werden.

3 Geschichte

3.1 Das GNU-Projekt

Das GNU¹-Projekt wurde 1984 von Richard M. Stallman² (im Allgemeinen einfach RMS genannt) mit dem Ziel gegründet ein komplett freies Betriebssystem samt Anwendungsprogrammen zu schreiben. Wobei *frei* hier nicht im Sinne von kostenlos, sondern im Sinne von Freier Software zu sehen ist, d.h. der Benutzer hat das Recht die Software zu verändern und zu verteilen. Um diese besondere Bedeutung von „frei“ zum Ausdruck zu bringen, wird gerne der erste Buchstabe groß geschrieben, es wird also von „Frei“ gesprochen, beziehungsweise im Englischen von „Free,, , im Gegensatz zu „free“. Zur kurzen und bündigen Erläuterung worum es bei Freier Software geht, kommt auch der Spruch: „free“ as in „free speech,“ not as in „free beer“ gerne zum Einsatz. Das GNU-System sollte dabei kompatibel zu Unix sein, aber es erweitern und verbessern, falls nötig.

Im Jahre 1985 hatte das GNU-Projekt schon eine Vielzahl an Tools, unter anderem den Texteditor Emacs³ einen yacc-kompatiblen Parser-Generator, sowie einen Compiler und einen Linker hervorgebracht. Zu einem kompletten System fehlte jedoch noch eine wichtige Komponente: Der Kernel. Die Rolle des Kernel sollte planmäßig vom *Hurd* übernommen werden, es kam aber anders und der Linux-Kernel wurde deutlich schneller benutzbar, der Hurd-Kernel ist es erst seit kurzem, sodass die meisten freien Systeme heute auf dem Linux-Kernel basieren, bzw. auf den größtenteils unabhängigen Entwicklungen von BSD (s.u). RMS gründete zur Unterstützung seiner Pläne die *Free Software Foundation*, kurz FSF.

3.2 Der Hurd-Kernel

Die Entwicklung des Hurd-Kernels wurde im Jahre 1986 begonnen. Zunächst hatte die FSF damit begonnen den TRINIX Kernel ihren Bedürfnissen anzupassen und Fehler zu beseitigen. Das wurde aber ein Jahr später zugunsten des Mach Kernels aufgegeben. Nachdem der Mach-Kernel 1990 unter einer freien Lizenz herausgegeben wurde, wurde damit begonnen den Hurd-Kernel auf Basis des Mach-Kernels zu entwickeln. Die Entwicklung ging jedoch re-

¹ GNU steht für *GNU's Not Unix* und ist ein rekursives Akronym

² <http://www.gnu.org/people/rms/>

³ Emacs ist ein in *Lisp* programmierbarer Texteditor. Durch seine Programmierbarkeit ist er nahezu unbegrenzt erweiterbar. Inzwischen gibt es für den Emacs Erweiterungen, die es erlauben ihn als wissenschaftlichen Taschenrechner (*calc*), als Mail- und News-Reader, als Datenbank oder gar als Shell-Ersatz (*eshell*) zu benutzen.

lativ schleppend voran, denn im Gegensatz zu Linux setzt der Hurd-Kernel nicht auf bewährte Technologien, sondern betritt in vielen Gebieten Neuland. So ist der GNU Hurd kein monolithischer Kernel, sondern baut auf einem Mikrokern auf.

Der GNU/Hurd-Kernel erreicht nun langsam auch ein Stadium, in dem er für viele Einsatzgebiete benutzbar wird und das Debian-Projekt arbeitet an einer auf Hurd basierenden Distribution – *Debian GNU/Hurd*.

3.3 Linux

Der Linux-Kernel entwickelte sich anders als das GNU-System. Er war nicht mit dem Ziel gestartet worden ein von vielen benutzbarer Kernel zu werden oder gar ein komplettes Betriebssystem, er war auch nicht durch ethische oder philosophische Gesichtspunkte motiviert. Statt dessen entstand er vielmehr durch die teils chaotische Zusammenarbeit von vielen Interessierten unter der Führung des finnischen Informatik-Studenten Linus Torvalds⁴.

Das Ganze begann im April 1991, als Linus anfang an einem neuen unixartigen Betriebssystem zu arbeiten. Das Projekt war anfangs nur dazu gedacht den 386er näher kennenzulernen, und so wurde auf Portabilität und ähnliche Gesichtspunkte erstmal wenig Wert gelegt. Als Ausgangspunkt für das Projekt diente *Minix*, ein kleines, kostenpflichtiges Unix-Betriebssystem, das vor allem für den Unterricht gedacht war. Einige Monate später, im Oktober 1991, kam dann die erste öffentliche Linux-Version heraus, 0.02. Sowohl die GNU Bash als auch der GNU-Compiler waren portiert worden. In den Monaten darauf folgten weitere Versionen und weitere Leute schlossen sich an. Es gab auch einiges an Bibliotheken, die für Minix⁵ geschrieben worden waren und jetzt auf Linux portiert wurden. Mit 0.11 kam dann die erste Version heraus, die eigenständig war. Alle vorigen brauchten noch ein Minix, um sie zum Laufen zu bringen.

Mit der Version 0.12 wurde Linux dann unter die GNU GPL gestellt, da die vorherige Lizenz zu restriktiv war. Sie erlaubte zum Beispiel nicht, mit Linux Geld zu verdienen. Bereits 1992 wurde von *SuSE* die erste kommerzielle Linux-Distribution erstellt. Zwei Jahre später folgte dann die Firma *Red Hat*. 1993 wurde das Debian Projekt⁶ gestartet, das es sich zum Ziel gesetzt hatte eine freie Linux-Distribution zu entwickeln. Heute gibt es eine Vielzahl von Linux-Distributionen für die verschiedensten Einsatzgebiete.

⁴ <http://www.cs.helsinki.fi/~torvalds/>

⁵ <http://www.cs.vu.nl/~ast/minix.html>

⁶ <http://www.debian.org/>

3.4 BSD

Die erste BSD-Variante (*Berkeley Software Distribution*) wurde 1978 an der Berkeley Universität von Kalifornien⁷ basierend auf UNIX V6 entwickelt. Unix war 1969 von AT&T in den Bell Labs entwickelt worden. Universitäten waren in der Lage Unix für sehr wenig Geld zu kaufen und erhielten sowohl den gesamten Sourcecode, als auch das Recht diesen beliebig zu verändern.

Da bei AT&T jedoch zunehmend kommerzielle Interessen in den Vordergrund rückten und somit Unix ab 1984 unter einer rein kommerziellen Lizenz vertrieben wurde, versuchte man den ursprünglichen Code durch eigenen zu ersetzen, was auch 1993 mit BSD 4.4 endgültig erreicht worden war.

Bereits 1991 hatten William und Lynne Jolitz BSD auf den 386 portiert und diese Variante 386BSD genannt. Die drei heutigen BSDs basieren alle auf 386BSD zusammen mit einem Teil des Codes aus BSD 4.4. FreeBSD wurde ins Leben gerufen, um das Installieren auf 386-basierten Maschinen zu erleichtern. Mit der Entwicklung von NetBSD wurde etwa zur selben Zeit begonnen mit dem Ziel BSD auf eine möglichst große Zahl von Rechnerplattformen zu portieren. OpenBSD wurde dann 1996 von dem NetBSD Projekt abgespalten, da einige Entwickler der Meinung waren, die Sicherheit würde bei NetBSD nicht ausreichend beachtet. OpenBSD wurde seitdem speziell in Bezug auf Systemsicherheit weiterentwickelt. Die drei BSD Varianten bieten also zu dem GNU System, beziehungsweise zu Linux eine ebenso freie Alternative, die allerdings nie so recht die Bekanntheit von Linux erreicht hat, die Gründe hierfür mögen wohl hauptsächlich im geschlosseneren Entwicklungsmodell von BSD liegen.

3.5 Apache

Der *Apache Webserver*⁸ ist neben GNU/Linux und BSD eines der erfolgreichsten und bekanntesten Open-Source-Projekte überhaupt. Der Name Apache geht auf die Bezeichnung „A Patchy Server“ zurück, da Apache aus einer Sammlung von Patches für den NCSA-(National Center for Supercomputing Applications)-Webserver entstanden ist. Die Apache Group wurde im Februar 1995 von acht Mitgliedern gegründet. Bereits im April 1995 wurde dann die erste öffentliche Version (0.6.2) des Apache-Servers vorgestellt. Obwohl bereits diese frühe Version vielfach eingesetzt wurde, war den Entwicklern klar, dass die gesamte Codebasis überarbeitet werden musste, wollte man die hohen Ziele, die man sich für die Entwicklung des Apache gesteckt hatte, erfüllen. Daher wurde von Robert Thau eine vollkommen neue Serverarchi-

⁷ <http://www.berkeley.edu/>

⁸ <http://www.apache.org/>

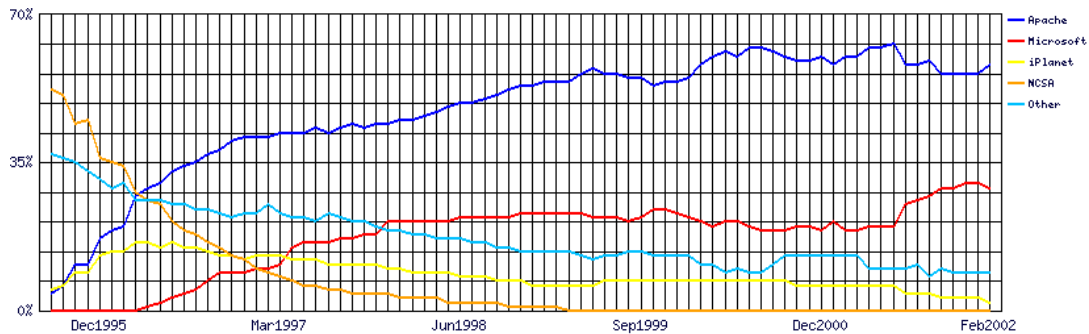


Abbildung 1: Marktanteile verschiedener Webserver

tektur entwickelt, die seit der Version 0.8.8 Basis des Apache ist. Die Version 1.0 wurde am ersten Dezember 1995 veröffentlicht und war so erfolgreich, dass Apache innerhalb eines Jahres nach der Gründung der Apache Group bereits den NCSA-Webserver als den am häufigsten eingesetzten Webserver überholt hatte und die Position bis heute halten konnte, wie die Netcraft-Studien⁹ belegen (Abbildung 1). War die Apache Group anfänglich nur gegründet worden, um den Webserver der NCSA weiterzuentwickeln, so haben sich hieraus bis heute eine ganze Reihe von Projekten entwickelt, die das Ziel verfolgen, nicht nur einen Webserver, sondern auch Software zu entwickeln, die benötigt wird um den Server möglichst vielseitig einsetzen zu können. Aus der Apache Group entstand im September 1999 die Apache Software Foundation, die es sich zum Ziel gesetzt hatte die Apache Open-Source-Projekte sowohl finanziell als auch rechtlich zu unterstützen. Einige Mitglieder der Apache Software Foundation vertreten bekannte Firmen wie z.B. Red Hat, IBM oder Sun Microsystems. Momentan befindet sich die Version 2.0 des Apache Webservers in der Entwicklung, die neben vielen Detailverbesserungen vor allem die Skalierbarkeit erhöhen soll, sodass Apache besonders auf großen Systemen besser eingesetzt werden kann. Diese neue Version wird wohl die Vormachtstellung des Apache als führenden Webserver weiter festigen. (hs)

4 Lizenzen für Freie Software

Wenn Programme als Open Source oder Freie Software veröffentlicht werden heißt das noch lange nicht, daß jeder damit machen darf was er will. Rechte und Pflichten werden für den Anwender in Lizenzen reglementiert. Alle Open-Source-Lizenzen haben per Definition durch die *FSF*¹⁰ (*Free Software*

⁹ <http://www.netcraft.com/survey/>

¹⁰ FSF: <http://www.fsf.org/>

Foundation) und die *OSI*¹¹ (*Open Source Initiative*) die folgenden Grundrechte in sich verankert:

- Recht zur Einsicht in die Quelltexte des Programms
- Recht auf Änderung der Quelltexte zu eigenen Zwecken
- Recht auf Weitergabe der veränderten Quelltexte

Der Sinn hinter diesen grundlegenden Rechten liegt darin, dass Entwickler auf der Arbeit von anderen aufbauen können. Leute mit guten Ideen und Programmiererfahrung müssen so nicht komplett von vorne anfangen, sondern haben bereits ein Fundament, auf das sie eigene Ideen aufsetzen können.

Als Motiv für die Lizenzierung von Open-Source-Programmen gibt es aus Entwicklersicht verschiedene Aspekte. Zum einen wäre da die Anerkennung, die ein Autor sich erhofft, wenn sein Programm publik wird. So darf BSD (Berkley Software Distribution) lizenzierte Soft nach belieben verändert und/oder verteilt werden mit der Verpflichtung die Namen der ursprünglichen Autoren zu nennen.

Ein zweites Motiv ist der Schutz vor „Raubbau“ an Open-Source-Software durch kommerzielle Produkte. Als Beispiel ist hier die *GPL* (*GNU General Public License*) zu nennen, die verfügt, dass ein Programm, das *GPL*-lizenzierten Code enthält, wieder unter die *GPL* gestellt werden muss. Es ist so nicht legal Code zu klauen und als „eigenes“ proprietäres Produkt zu verkaufen. Freier Code bleibt dadurch frei.

Drittens möchten manche Programmierer sich davor schützen, wenn Sie die Software schon frei zur Verfügung stellen, dass bei möglichen Schäden, die die Software verursachen könnte, sie auch noch dafür verantwortlich gemacht werden. Fast jede Lizenz enthält deswegen einen Haftungsausschluss des Autors für mögliche Konsequenzen, die der Einsatz der Software mit sich bringt.

Als letztes bestehen manche darauf, dass abgeänderte Versionen der Software nicht den „offiziellen“ Namen tragen dürfen, um Verwechslungen zu vermeiden oder damit nicht viele zueinander inkompatible Versionen entstehen. Die Apache-Lizenz legt hierauf einen Schwerpunkt.

Da es eine Unzahl von Lizenzen gibt soll hier nur auf die wichtigsten eingegangen werden.

¹¹ OSI: <http://www.opensource.org/>

4.1 GPL – GNU General Public License

Die bekannteste unter den Open-Source-Lizenzen ist wohl die *GNU* General Public License, kurz *GPL*¹². Sie gewährt dem Anwender die Freiheit ein Programm sowohl modifiziert als auch unverändert beliebig weiterzugeben. Gleichzeitig wird allerdings gefordert, dass das weitergegebene Programm auch im Quelltext beiliegt, oder auf Wunsch zugänglich gemacht wird und ferner das Programm wieder unter der *GPL* stehen muss. Eine Klausel beinhaltet einen Haftungsausschluss des Autors gegenüber Schäden. Laut der *GPL* ist es nicht verboten das Programm zu verkaufen, wobei allerdings der Verkauf nicht mit einer Einschränkung der Rechte einhergehen darf. Das macht es schwierig, *GPL*-Programme ohne sonstige Leistungen, wie z. B. Support, Garantie o. ä. zu verkaufen, da derjenige, der es gekauft hat, es kostenlos weitergeben darf. Auch wenn nur Teile der Sourcen genutzt werden, muss nach der *GPL* die gesamte Software wieder unter die *GPL* gestellt werden. Dieses Abfärben, welches von Microsoft auch gerne als Virus-Effekt bezeichnet wird, erklärt neben den offensichtlichen Vorteilen auch die weite Verbreitung dieser Lizenz.

4.2 LGPL – Lesser General Public License

Die *LGPL*¹³ wurde geschaffen um ein Problem zu umgehen, welches durch die *GPL* entsteht. Und zwar gibt es *GNU-C*-Bibliotheken, die Code enthalten, der von verschiedenen Programmen genutzt werden kann. Dies ist vorteilhaft, weil Programmierer nicht bekannte Routinen neu schreiben müssen, wenn sie ein bestimmtes Vorhaben umsetzen wollen, sondern sie können in ihren Anwendungen gegen diese Bibliotheken linken, das heißt eine Referenz auf einen Algorithmus der Bibliothek setzen, um Arbeit und Plattenplatz zu sparen. Die *GPL* würde aber nun verlangen, dass die ganze Anwendung, die Code von einer Bibliothek gebraucht, als abgeleitetes Produkt auch wieder unter der *GPL* stehen müsste. Es wäre so kaum möglich kommerzielle Produkte, die gegen die Bibliothek gelinkt sind, zu schreiben. Hier hilft nun die *LGPL* aus, die fast genauso aufgebaut ist, wie die *GPL*, nur mit der Erlaubnis abgeleitete Werke auch ohne Quelltext und ohne Recht auf Weitergabe zu vertreiben. Es muss dem Anwender allerdings erlaubt und möglich sein die Software gegen eine veränderte Bibliothek zu linken.

¹² GPL: <http://www.gnu.org/copyleft/gpl.html>

¹³ LGPL: <http://www.gnu.org/copyleft/lesser.html>

4.3 BSD – Berkeley Software Distribution

Bei der BSD-Lizenz¹⁴, die neben der GPL wohl die zweithäufigste ist, muss der Lizenztext in den Quellen oder Binaries erhalten bleiben. Der ursprüngliche Entwickler muss bei der Weitergabe eines veränderten Programmes lobend erwähnt und somit anerkannt werden.

Hier ist es im Unterschied zur GPL allerdings nicht notwendig, dass die Quelltexte eines modifizierten Programms weitergegeben werden. Einer Vermischung mit kommerzieller Software ist so durchaus möglich. Eine seit dem 22. Juli 1999 gestrichene „Advertising“-Klausel besagte, dass bei der Werbung für ein Produkt, das BSD lizenzierten Code enthält, alle Programmierer genannt werden müssen, die an der Entwicklung maßgeblich beteiligt waren. Die Streichung in neueren Versionen der BSD-Lizenz erklärt sich dadurch, dass zur einwandfreien Bewerbung einer *NetBSD-CD* inzwischen schon fast 100 Entwickler genannt werden müssten. Man stelle sich eine Werbung für dieses Produkt in einer Computerzeitschrift mit Beachtung der entsprechenden Klausel vor.

4.4 Apache

Der Webserver Apache hat eine eigene¹⁵, leicht veränderte BSD-Lizenz. Der Quelltext kann beliebig verändert und verbreitet werden, nur die Autoren müssen lobend erwähnt werden. Bei der Verbreitung von veränderten Sourcen darf ein abgeleitetes Produkt nicht „Apache“ als Bestandteil des Namens haben. Der Sinn hinter dieser letzten Regel ist der Wunsch der Programmierer Erweiterungen und Neuerungen von anderen möglichst in ihren Apache zu integrieren und so eine Aufspaltung in viele Ultra- und Mega-Apaches zu verhindern.

4.5 QPL – Q Public License

Die Q Public License¹⁶ die zur Lizenzierung der Qt-Bibliothek genutzt wird, hat inzwischen an Bedeutung verloren. Die Bibliothek enthält diverse GUI- (Graphical User Interface)-Routinen, die beispielsweise vom KDE (*K Desktop Environment*) genutzt werden. Die QPL fordert, dass Quelltextveränderungen getrennt von der Original-Bibliothek gehalten und mindestens dem Hersteller Trolltech zugänglich gemacht werden müssen. Aber inzwischen lizenziert Trolltech auch eine „Free Edition“ von QT unter der GPL (siehe in Ka-

¹⁴ BSD: <http://www.opensource.org/licenses/bsd-license.html>

¹⁵ Apache: <http://www.apache.org/LICENSE.txt>

¹⁶ QPL: <http://www.trolltech.com/developer/licensing/qpl.html>

pitel 4.1). Um dem ganzen noch einen drauf zu setzen, existieren auch noch kommerzielle Lizenzen, die man für proprietäre Produkte benutzen muss. Bei der Linux-Version der Bibliothek kann man zwischen freier und kommerzieller Lizenz aussuchen, während für Portierungen nach Windows bzw. Mac OS nur die kommerzielle Variante existiert. (ph)

5 Funktioniert Freie Software überall?

Wenn man so Freie Software betrachtet und schaut, was eine normale GNU/Linux Distribution alles leisten kann, stellt sich die Frage, ob sich dieses funktionierende „Konzept“ auf alle Bereiche der Software ausweiten lässt. Der folgende Abschnitt wird mehrere Bereiche von Software beleuchten und Prognosen aufstellen, wie sich Freie Software dort verhalten und entwickeln wird.

5.1 Office, E-Mail und Co.

Für den Otto Normal-PC-Benutzer ist zum Arbeiten im Wesentlichen eine Textverarbeitung, eine Tabellenkalkulation, ein Internetbrowser, ein E-Mail-Programm und noch ein Grafikprogramm wichtig, zumindestens wenn man mal danach geht, was bei einem normalen PC so alles vorinstalliert ist. Wenn man eine GNU/Linux-Distribution heute betrachtet, stellt man schnell fest, dass die wesentlichen Bedürfnisse schon recht gut abgedeckt sind. Es gibt mit StarOffice eine komplette Office-Suite, die zwar Closed-Source ist, aber deren Nachfolger Open Office unter der GNU GPL steht und damit Freie Software ist. Wenn Open Office also in ein paar Monaten veröffentlicht wird, steht eine freie Office Suite zur Verfügung. Daneben gibt es alternativ noch *LaTeX*, *KOffice*, *Abiword* und *Gnumeric*, die ebenfalls die Office-Bedürfnisse des Nutzers bedienen können. Bei E-Mail sieht es prinzipiell ähnlich aus, es gibt diverse freie E-Mail-Clients, wie *KMail*, *mutt* und *Evolution*. Als Internet-Browser stehen mit *Konqueror* und *Mozilla* zwei freie Alternativen zu dem Microsoft Internet Explorer zur Verfügung. Ebenso sieht es bei der Grafikbearbeitung aus, hier steht mit Gimp¹⁷ eine leistungsfähige Alternative zu Adobe Photoshop zur Verfügung.

Im Heimbereich ist also die Basisversorgung durch Freie Software gegeben, wenn auch manche der Office-Programme noch relativ jung sind. Probleme dürften hier teilweise aber noch Finanzverwaltungs-Software und Ähnliches sein, aber auch hier gibt es mit *GnuCash* eine freie Alternative.

¹⁷ <http://www.gimp.org>

5.2 Server-Software

Im Gegensatz zu Office-Software, bei der die Freie Software der kommerziellen noch ein klein bisschen hinterherhinkt, ist das bei Web-Server-Software eher genau umgekehrt. Der Apache, der der weltweit meist genutzte Webserver ist (s. Abb. 1, S. 7), war schon immer Freie Software. PHP, Zope und ähnliche fürs Web gemachte Skriptsprachen sind ebenfalls Freie Software. Ähnlich sieht es im Bereich von Mail-Servern aus, mit Sendmail, Postfix und Exim steht hier hinreichend viel und vor allem auch viel eingesetzte Freie Software zur Verfügung. Proprietäre Software hinkt hier teilweise hinterher, nicht zuletzt weil im Bereich des Internets Freie Software und offene Standards schon immer eine antreibende Kraft waren.

5.3 High-End-Software

Mit High-End-Software ist Software gemeint, die nicht für den Privatkunden gemeint ist, sondern vor allem, bzw. nur, im *Business-to-Business*-Bereich genutzt wird. Als Beispiel wären hier 3D-Software-Tools wie *Maya* oder *Lightwave3D* anzuführen, wie sie in diversen Filmproduktionen genutzt werden. In diesem Bereich hat Freie Software in der Regel kaum etwas zu bieten, da hier schlichtweg nur geringer Bedarf beim Privatkunden herrscht solche Software einzusetzen, bzw. für einen einzigen Programmierer ist solche Software kaum zu entwickeln. Für Firmen besteht ebenso wenig Anreiz solche Produkte als Open Source frei zu geben. Es ist also nicht davon auszugehen, dass sich daran in nächster Zukunft etwas ändern wird, letztendlich weil der Bedarf einfach nicht da ist. Allerdings sollte auch erwähnt werden, dass Freie Software im Filmbereich durchaus eine Rolle spielt, zwar nicht so sehr auf dem Schreibtisch, sondern eher im Serverraum, wo oft einige dutzend oder hundert Linuxrechner für das Berechnen von Computeranimationen benutzt werden, so geschehen zum Beispiel für *Titanic* ([1]) oder *Final Fantasy*.

5.4 Spiele

Spiele sind letztendlich ein weiterer wichtiger Teil an Software, die der Otto Normalbenutzer gerne hätte, im Gegensatz zu Office-Software gibt es hier bei Freier Software aber keine großartigen Alternativen zu proprietären Spielen. Woran liegt das? Im Wesentlichen liegt es wohl daran, dass Freie Software dort erfolgreich ist, wo es um langlebige Software geht, also Software die noch in zwei oder fünf Jahren nützlich ist. Bei Spielen ist das anders, Spiele sind im Wesentlichen *Wegwerfprodukte*, das heißt in zwei Jahren interessiert sich kaum noch einer für ein Spiel von heute. Außerdem stellt ein Spiel eine recht

hohe Anforderung nicht nur an die Programmierer, sondern auch vor allem an die Künstler, denn ein Spiel besteht zu großen Teilen aus Grafik und Sound, Programmcode spielt oft nur eine Nebenrolle, wenn auch diese ziemlich groß ist. Ein durchschnittliches Spiel hat nämlich etwa 250 000 oder mehr Zeilen Code und kann damit problemlos mit den größeren freien Software-Projekten mithalten (Apache, Gimp und Co.).

Bei Spielen hat Freie Software des Weiteren noch das Problem, dass Spiele in der Regel von geschlossenen Teams entwickelt werden, was auch nötig ist, um Konsistenz in das Spieldesign zu bringen, dass Freie Software in der Regel verteilt entwickelt wird, verkompliziert die Sache noch zusätzlich.

Des Weiteren mangelt es im Bereich der Freien Software auch noch an wichtigen Tools, die für die Entwicklung von modernen 3D-Spielen nötig sind, im Wesentlichen sind das Tools zur Modellierung von 3D-Modellen. Hier könnten eventuell *Blender*, das jetzt nach dem Bankrott von *NaN*¹⁸ eventuell Freie Software wird, oder *PrettyPolyEdit*¹⁹ eine Alternative werden, aber bis diese Programme mit den kommerziellen mithalten können wird noch einiges an Zeit vergehen.

Aber es besteht zumindestens in einigen Bereichen Hoffnung, dass Freie Software eine Alternative werden kann. Als Beispiel wären hier die Massive Multiplayer Online RPG (kurz MMORPG) zu erwähnen. Deren Entwicklungsmodell scheint deutlich näher an Freier Software zu sein, als das bei herkömmlichen Spielen der Fall ist. Vor allem sind MMORPGs keine Wegwerfprodukte, sondern bieten über einen längeren Zeitraum eine immer größer werdende und erweiterbare Spielewelt. Das *WorldForge*²⁰ bietet solch eine Plattform, auf der zukünftige MMORPGs entstehen könnten.

Ein weiterer Punkt, in dem Freie Software eine Rolle spielen könnte, ist nicht so sehr bei den Spielen selbst, sondern bei den darunterliegenden Libraries und Engines. Engines und Libraries sind letztendlich der Teil eines Spiel, der nicht nach dem Release weggeworfen wird, sondern eventuell in weiteren Spielen verwendet werden kann. Hier könnten freie Engines also in der Tat für einige Firmen zu einer Alternative werden. Wobei allerdings auch eine Engine in der Regel nicht viel länger als zwei Jahre hält, danach ist sie veraltet und würde von der Hardware im Grunde überholt. Game Libraries halten in der Regel aber noch ein gutes Stück länger, da sie weniger auf eine konkrete Hardware zugeschnitten sind. Als Beispiele von Freien Game-SDKs

¹⁸ <http://www.blender3d.com/>

¹⁹ <http://pretypoly.sourceforge.net/>

²⁰ <http://www.worldforge.net/>

wären hier *CrystalSpace*²¹, *ClanLib*²², *SDL*²³ und *plib*²⁴ anzuführen, die auch teilweise schon in kommerziellen Produkten genutzt werden.

Eine weitere Thematik im Bereich der Spiele, die eine Hoffnung für Freie Software sein könnte, ist das Mod-Development. Damit bezeichnet man Modifikationen zu fertigen Spielen. Zum Beispiel neue Waffen, Models oder Levels für einen Ego-Shooter, neue Flugzeugtypen für einen Flugsimulator oder Ähnliches. In den letzten Jahren hat sich die Mod-Szene enorm vergrößert und bei neueren Spielen wird inzwischen auch sehr viel Wert darauf gelegt, dass sie relativ leicht zu modifizieren sind. Die Mods werden dabei von kleinen, freiwilligen Teams zusammengebastelt und kostenlos verteilt. Die Entwicklung geschieht also in der Regel ähnlich wie bei Freier Software, ohne dass zwangsläufig kommerzielles Interesse dahinterstehen muss.

So gesehen heißt also der Mangel, der im Augenblick an Freien Spielen herrscht, nicht zwangsläufig, dass es nie welche geben wird, tatsächlich zeigt er eher, dass Freie Software in diesem Bereich noch nicht besonders weit gekommen ist und sich nötige Veränderungen erst in den letzten Jahren ergeben haben, wie z. B. Unterstützung für 3D-Grafikkarten und 3D-Editoren, die zwingende Voraussetzungen für Spiele sind. Es dürften also vermutlich mit der Zeit auch ein paar freie Spiele zustande kommen, aber das hängt wohl nicht zuletzt von den Nutzerzahlen von Freier Software ab, denn momentan benutzen die meisten potentiellen Spieleentwickler wohl noch Windows, da GNU/Linux für sie sowohl aus Entwickler- also auch aus Spielersicht noch nicht interessant genug ist. Es herrscht hier also ein Henne-Ei Problem – ohne Spieler kommen keine Entwickler und ohne Entwickler kommen keine Spieler zu GNU/Linux.

(ir)

6 Freie Software an Schulen

Die bekannten Argumente für Freie Software gelten natürlich auch für den Einsatz in Schulen: Geringe Kosten, hohe Sicherheit, leichte Erweiterbarkeit, vor allem ersteres. Aber kommt es wirklich so stark darauf an? Für eine Schule, die sich darum kümmert, ist es recht einfach möglich „Microsoft-Partnerschule“ zu werden. Somit erhält man alle wichtigen Microsoft-Programmpakete zum kostenlosen Unterrichtseinssatz, selbst zu Hause dürfen die Lehrer die Software nutzen.

Wichtiger scheinen andere – eher pädagogische – Gründe zu sein, die für

²¹ <http://crystal.sourceforge.net/>

²² <http://www.clanlib.org/>

²³ <http://www.libsdl.org/>

²⁴ <http://plib.sourceforge.net/>

Freie Software sprechen. Hat der Lehrer die Möglichkeit, eher theoretische Probleme der Informatik, wie z.B. Sortieralgorithmen oder die „binäre Suche“ an einem realen Programm vorzuführen, das von diesen Techniken Gebrauch macht, sollte es ihm eher gelingen, das Schülerinteresse zu wecken.

Aber auch die andere Richtung ist möglich: Selbsterstellte Programme finden durch eine passende Lizenz den Weg in die Öffentlichkeit. Interessant wird es, sobald das Programme tatsächlich für andere Nutzen hat. Beinahe zwangsläufig wird es woanders genutzt werden und erzeugt Rückmeldungen, Anfragen, die beantwortet werden müssen, was zur Förderung eines Verantwortungsbewusstseins beiträgt. Mit viel Glück erreicht das Projekt eine große Verbreitung, sodass die Anfragen (wohl meistens in Form von E-Mails) in englischer Sprache ankommen – schon wird der Unterricht fächerübergreifend.

Leider sind diese Ideen größtenteils Utopie; dass es aber dennoch funktioniert, zeigt das Projekt von Schülern von Leistungskursen zweier Gymnasien in Münster und der Universität Münster, die in gemeinsamer Arbeit ein funktionsfähiges Rastertunnelmikroskop entworfen und gebaut haben. Die selbst erstellte Software, Schaltpläne und Bauteilelisten sind online unter einer freien Lizenz verfügbar²⁵. Für „Forschungsprojekte“ wie diese ist eine freie Lizenz optimal: Es geht nicht darum, den Geld-Gewinn zu maximieren, sondern den Wissensgewinn. Indem die Software und Baupläne freigegeben werden, erhalten die Schüler und Uni-Mitarbeiter das bestmögliche Feedback von den Benutzern.

(mm)

7 Gefahren für freie Software

7.1 Patente

Softwarepatente stellen in immer höherem Maße eine Gefährdung für die Entwicklung neuer Programme dar. Innovationen auf dem Softwaresektor basieren heute selten auf komplett neuen Verfahren, sondern auf der geschickten Verknüpfung bereits vorhandener Algorithmen in einem neuen Projekt. Damit dies funktioniert, ist der Zugriff auf ein großes Repertoire von Algorithmen von großer Wichtigkeit.

In den letzten Jahren hat leider die Praxis zugenommen (insbesondere in den U.S.A.), Patente auf Verfahren mit äußerst geringer Innovationshöhe zu erwerben.²⁶ Neuentwicklungen werden so insbesondere in mittelständischen Unternehmen, in denen der Schwerpunkt der Entwicklung stattfindet,

²⁵ <http://sxm4.uni-muenster.de>

²⁶ Als recht bekanntes Beispiel sei hier Amazons One-KlickTM-Patent genannt.

erschwert. Selten können sie sich die Lizenzierung eines Verfahrens leisten, bzw. verfügen nicht über das finanzielle Polster, in einer von einem großen Konzern gegen sie angestregten Patentklage die Trivialität des Patents aufzuzeigen zu können.

Ironischerweise ist die Problematik bei Freier Software, die ja grundsätzlich quelloffen ist, noch verschärft. Es ist hier für Patentinhaber besonders leicht nachzuprüfen, ob ein patentiertes Verfahren genutzt wird. Und da Freie Software an sich für die Programmierer keinen direkten Gewinn abwirft, können sie natürlich auch keine Lizenzkosten für patentierte Verfahren bezahlen.

Aus den genannten Gründen sind daher die Forderungen kommerzieller Interessenverbände als auch die von Interessengemeinschaften im Bereich Freier Software²⁷ nachvollziehbar, die eine Erhöhung des Patentierungsniveaus für sinnvoll halten und mit ihren Einwänden besonders die bevorstehende Änderung des europäischen Patentrechtes beeinflussen wollen.

7.1.1 Das europäische Patentrecht

Ende 2000 sollte über eine Änderung des europäischen Patentrechtes beschlossen werden, die u.a. auch die Streichung einer Klausel beinhaltet hätte, die besagt, dass Programme grundsätzlich nicht patentiert werden können (Artikel 52). Aufgrund der Wichtigkeit des Themas und kontroverser Meinungen hierzu wurde beschlossen, die Entscheidung zu vertagen und die Änderung zunächst in der EU zu diskutieren. Auch die Meinung der Öffentlichkeit wurde erbeten.

Bei der Auswertung (nachzulesen in [4] und [2]) waren zahlenmäßig ganz klar die Anhänger Freier Software im Vorteil. Unter Berücksichtigung des wirtschaftlichen Gewichts der Befürworter der Patentrechtsnovellierung (Große Firmen, Patentanwälte und Patentbehörden) fiel die Empfehlung aber natürlich zu Gunsten eines ausgeweiteten Patentrechts aus.

Im November 2001 wurde dann die Richtlinie eigenmächtig durch das EPA (Europäisches Patentamt) geändert ([5]). In der neuen Form ist jede Computer-implementierbare Erfindung patentierbar, die einen technischen Charakter hat – wobei es schwer sein dürfte ein Programm zu finden, das diese Anforderung nicht erfüllt.

Für die, die es sich leisten konnten, wird sich allerdings nicht viel ändern: Denn schon vorher war – wie in [3] gezeigt – die nicht-Patentierbarkeit von Software vom Gesetzgeber in den letzten Jahren ausgehebelt worden.

Gezeigt wird dies an der einfachen Tatsache, dass in den letzten Jahren eine große Zahl von Patenten in dieser Richtung erteilt wurden:

²⁷ Zum Beispiel LinuxTag e.V. oder Linux-Verband „Live“

Allein in der internationalen Patentklasse G06F, die fast ausschließlich Software betrifft, hat das EP von 1978 bis zum Mai 1999 insgesamt 7619 europaweite Patente erteilt. [3]

Problematisch auch hierbei: Um die Lücken in der Nicht-Patentierbarkeit zu finden²⁸, ist Zeit und Geld nötig – ein kleines Unternehmen wird sich ohne das Wissen leicht mit der Auskunft abspeisen lassen, dass Softwarepatente generell nicht möglich seien. Große Unternehmen können dadurch ihre Marktposition auf Kosten der Mitbewerber weiter ausbauen.

7.1.2 Patentpraxis in den U.S.A.

In den U.S.A. ist es grundsätzlich möglich Software zu patentieren. Die negativen Folgen dessen wurden in letzter Zeit immer offensichtlicher: Große Unternehmen verklagen andere auf Patentverletzungen. Je mehr Patente der Gegner hat, desto wahrscheinlicher ist es, dass sich unter ihnen ein gleichwertiges findet, das vom Kläger verletzt wird, sodass im besten Fall die Auseinandersetzung in einem Vergleich, d.h. Patentaustauschabkommen endet, der beide viel Zeit und Geld gekostet hat.

Die Zahl der Patentanwälte nimmt immer stärker zu und anstatt in die Entwicklung investieren zu können, müssen teure Rechtsabteilungen beschäftigt werden, die allein zur Verteidigung gegen Patentklagen da sind. Folge des Bestrebens, sich durch eine möglichst große Anzahl eigener Patente eine gute Ausgangsposition in Patentstreitigkeiten zu sichern, ist die immer stärkere Zunahme von Trivialpatenten, die insbesondere dann für Freie Software besonders gefährlich werden, wenn sie sehr allgemein gehalten sind.

7.1.3 Überlastete Patentämter

Welch absurden Ausmaße dieses Verhalten annehmen kann, konnte im Juli 2001 ein Mann aus Melbourne, Australien zeigen, der sich das *Rad* patentieren ließ ([6]). Hätte er tatsächlich versucht, Lizenzgebühren für seine „Erfindung“ einzutreiben, wäre er sicherlich nicht weit gekommen, da die nötige Erfindungshöhe nicht gegeben ist. Aber eins schafft er sehr deutlich zu demonstrieren: Die durch Überlastung verursachte Unfähigkeit der Patentämter, Erfindungen und Nicht-Erfindungen korrekt einzustufen zu können. Dieses Problem besteht hierzulande genau wie in Australien.

Übertragen wir das Problem nun auf Software. Der Australier beschrieb seine „Erfindung“ etwas umständlich, dies reichte aus, um das alleinige Verwertungsrecht an einem Gegenstand zu bekommen, das eigentlich Allgemeingut sein sollte. Wenn wir uns vorstellen, dass Software an sich zwar etwas

²⁸ Z. B. kann Hardware zusammen mit passender Software patentiert werden.

komplizierter aufgebaut ist als ein Rad, aber dennoch häufig nicht weniger triviale Bestandteile enthält, ist es dann nicht anzunehmen, dass hier erst recht viele Patente – versteckt hinter unnötig verkomplizierenden, aber gleichzeitig verallgemeinernden Formulierungen – unberechtigterweise vergeben werden? Ohne eine entscheidende Erhöhung der Kompetenz der Patentämter kann diese Frage sicherlich mit ja beantwortet werden.

Die Gefahr betreffend, dass ein Algorithmus aus einer freien Software von einem Benutzer oder Autor patentiert werden könne, hat die GPL glücklicherweise Vorkehrungen getroffen:

Preamble:

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

In [7] wird berichtet, wie eine Firma, die eigene Erweiterungen des unter GPL stehenden Linux-Kernels patentieren ließ, eingelenkt hat keine Lizenzkosten für die Verwendung ihres patentierten Verfahrens in GPL-Programmen zu verlangen.

7.2 Proprietäre Standards

Die Situation Softwarepatente betreffend könnte einmal zu einer Gefahr werden, aber es gibt bereits jetzt bewusst herbeigeführte Hindernisse großer Hersteller, die andere Entwickler (sowohl kommerzieller als auch freier Software) effektiv daran hindern konkurrenzfähige Produkte herzustellen. Ein Softwarehersteller vermarktet dazu ein Produkt, das nur mit Produkten desselben Herstellers kommunizieren kann. Die Dateiformate und Protokolle, die zum Datenaustausch verwendet werden, werden verschlossen gehalten. Daran ist nicht unbedingt etwas auszusetzen, allerdings sieht die Situation schon anders aus, wenn es so aussieht, als wäre das Dateiformat mit Absicht derartig angelegt, dass es auch mit großem Aufwand einer Konkurrenzfirma kaum zu entschlüsseln ist.

Das Dateiformat von *Microsoft Word* sei hier als prominentestes Beispiel angeführt. Dieses von vielen als eher minderwertiger Teil von Microsofts Office-Paket angesehenes Programm konnte durch Microsofts Monopolstellung in den Markt gedrückt werden und avancierte so zu einem fragwürdigen Quasi-Standard. Denn: Es gibt kaum Programme, die das umständ-

liche Word-Dateiformat lesen, geschweige denn schreiben könnten²⁹. Unternehmen, die häufig von außen Dokumente erhalten, gehen häufig den Weg des geringsten Widerstandes und benutzen lieber gleich selbst die Microsoft-Produkte – auch wenn die Textverarbeitung der Konkurrenz noch so gut sein mag, sie kann ja nicht mit den „Standard“-Dateien umgehen. Und wieder wurde ein eventuell besseres Produkt aus dem Markt gedrängt.

Freie Software hat hier ein Problem: Große Softwarehersteller haben es mit erheblichem finanziellen und zeitlichen Aufwand geschafft, die Verarbeitung proprietärer Standards in ihren Produkten zu ermöglichen, weil es für sie nötig war, um am Markt konkurrenzfähig bleiben zu können. Für freie Software besteht dieser Druck kaum: Benutzen nur noch 500 statt 1000 Leute das Programm, so ist dies zwar schade, aber für das Überleben des Programms prinzipiell unwichtig. Für die StarDivision GmbH arbeitete ein Jahr lang ein ganzes Team daran ihrem Produkt StarOffice das Öffnen und Schreiben von Microsoft-Word-Dokumenten beizubringen. Freien Entwickler fehlt die Zeit und sicherlich auch Motivation, so etwas umzusetzen.

Microsoft hat dies erkannt und in den sogenannten *Halloween-Papieren* neben anderen ethisch fragwürdigen Verfahrensweisen explizit darauf hingewiesen, dass proprietäre Standards in der Bekämpfung von freier Software zu evaluieren seien. (mm)

8 Businessmodelle

Ganz ohne Geld funktioniert Freie Software letztendlich auch nicht, irgendwer muss schließlich die Distributionen, die man am Ende im Laden kaufen kann, zusammenpacken und damit auch Geld verdienen, damit sich das Ganze auch lohnt. Im Folgenden werde ich einen kleinen Überblick darüber geben, wie Firmen mit Freier Software Geld verdienen können und in welchen Bereichen das gut funktioniert und in welchen eher nicht. Die Distributoren sind nämlich nicht die einzigen, die mit Freier Software Geld verdienen.

8.1 Distributoren

Wie erwähnt sind die Distributoren die ersten, die ins Auge fallen, wenn es ums „Geld machen“ mit Freier Software geht. Nun, was tun die Distributoren? In erster Linie bündeln sie die Software, die frei im Internet verfügbar ist, überdies schauen sie auch noch, dass die ganze gebündelte Software ordentlich zusammenarbeitet und patchen sie entsprechend, wenn das nicht der

²⁹ Sogar *Word* selbst hatte in Einzelfällen Probleme Dateien aus einer älteren Word-Version zu schreiben bzw. zu lesen.

Fall ist. Neben den Paketen, also der fertig gepackten und gepatchten Software, schreiben die Distributoren aber auch selber Software, so schreiben sie z. B. Tools zum (halb)automatischen Installieren von GNU/Linux oder Tools zum Verwalten desselben. Außerdem sind die meisten Distributoren auch erheblich an der Treiber-Entwicklung beteiligt, da sie letztendlich ein großes Interesse daran haben, dass die verkauften Distributionen auf allen Rechnern problemlos zum Laufen zu bringen sind und nicht etwa an einer nicht unterstützten Grafikkarte scheitern. Die Ergebnisse dieser Arbeit werden dann in der Regel als Freie Software veröffentlicht. Geld wird nun eingenommen, indem die Distributionen verkauft werden, wie andere Software auch.

Neben dem Verkaufen haben die Distributionen aber auch noch eine andere, vermutlich deutlich ergiebigere, Einnahmequelle. Sie leisten nämlich Support für diverse Firmen und betreuen die dortigen GNU/Linux-Installationen.

8.2 Support

Support ist letztendlich eine wichtige Dienstleistung, wenn es um Software geht und gerade hier bietet Freie Software eine gute Möglichkeit Geld zu machen. So können sich Firmen zum Beispiel um GNU/Linux-Installationen kümmern oder selbige einrichten. Allerdings funktioniert das Ganze in der Regel nur ordentlich, wenn man schon auf fertiger Freier Software aufbaut. Neue Freie Software wird beim Support nicht, bzw. nur in kleinen Rahmen geschrieben, wie z. B. Bugfixes und kleinere Features. Es wird beim Support also das Geld hauptsächlich durch eine Dienstleistung erbracht und nicht so sehr durch die Software.

8.3 Kleine Unternehmen

Eine ganz andere Art, mit Freier Software Geld zu verdienen, kann von kleineren Unternehmen, die Software herstellen, praktiziert werden. Firmen, die Spezialsoftware auf Anfrage erstellen, und nicht so sehr davon leben eine einzige Software millionenfach zu verkaufen, können durch Freie Software schlicht und ergreifend mehr Service bieten, bei gleichzeitiger Kosteneinsparung. Sie können nämlich einerseits auf vorhandener Freie Software aufbauen und so eventuell bessere Lösungen abliefern oder sie können dem Kunden die Freie Software als Vorteil verkaufen, da der Kunde nicht von der Firma, die die Software erstellt hat abhängig ist, sondern diese bei Bedarf jederzeit wechseln kann. Das Ganze kann vor allem funktionieren, da es sich um Spezialsoftware handelt, die nur eine geringe Anzahl von möglichen Kunden

hat, im Idealfall nur einen einzigen, nämlich den, der den Auftrag für diese Software gegeben hat.

8.4 Subscription-Modell

Ein ganz anderes Modell, aus Freier Software Geld zu machen, wird im Augenblick von Transgaming³⁰ versucht. Dort wird kein fertiges Produkt verkauft oder gar eines auf Auftrag hin erstellt, stattdessen wird dort versucht eine Emulation für DirectX zu schreiben; diese würde es möglich machen, dass viele herkömmliche Windows-Spiele direkt unter GNU/Linux laufen könnten. Es würde also letztendlich Windows für einen großen Anteil an Benutzern überflüssig machen, die heute noch Windows als Zweit-Betriebssystem für Spiele benutzen. GNU/Linux selbst ist nämlich für die Spiele-Produzenten als Markt nicht interessant genug, da die Verbreitung dafür noch nicht groß genug ist, bzw. da selbst viele GNU/Linux Benutzer immer noch ein Windows als Zweit-Betriebssystem haben. Die Software, die Transgaming herstellt, wäre also für eine breite Nutzerschicht, letztendlich alle PC-Spieler, interessant. Nun, wie funktioniert das Businessmodell nun? Das ist relativ einfach, man meldet sich bei Transgaming an und bezahlt einen monatlichen Beitrag von 5 Dollar. Für diesen Betrag bekommt man dann Zugriff auf vorkompilierte Pakete der Software und bekommt insbesondere ein Stimmrecht. Mit diesem Stimmrecht kann man dann darüber abstimmen, an welchem Problem als nächstes gearbeitet werden soll, hier also insbesondere die Kompatibilität zu einem bestimmten Spiel. Darüber hinaus bekommt man den Quellcode der Software auch ohne den Betrag bezahlt zu haben (dieser ist zwar im Moment *keine* Freie Software, da er nicht kommerziell vertrieben werden darf, aber modifizieren desselben ist erlaubt), allerdings hat man dann kein Stimmrecht und muss sich den Source selber kompilieren. Wenn eine bestimmte Anzahl an Nutzern vorhanden ist, soll die Software auch komplett Freie Software werden, die augenblicklichen Beschränkungen dienen im Wesentlichen dazu, erstmal die User an sich zu binden und eine Nutzerbasis aufzubauen. Die Chancen, dass das Ganze funktionieren wird, stehen nicht schlecht. Allerdings wurden bis jetzt noch keine Nutzerzahlen angegeben, das soll erst gegen Ende März 2002 passieren. Also weniger konkret gesprochen zahlen die Leute hier für die Arbeitskraft der Programmierer und weniger für ein fertiges Produkt, sie können so natürlich auch in gewisser Weise das fertige Produkt beeinflussen.

Im konkreten Fall könnte das Ganze funktionieren, da WineX ein großes Publikum anspricht und der Erfolg relativ gut sichtbar ist, da Zugriff auf

³⁰ <http://www.transgaming.com>

den Sourcetree der Entwickler problemlos möglich ist. Die Frage ist nun, ob das auch für andere Software funktionieren könnte? Also dass User direkt an die Entwickler bezahlen, ohne Umweg über Distributoren, Publisher oder Ähnliches. Im Wesentlichen gibt es da noch ein Problem und das ist das Bezahlen. Fünf Dollar sind zwar nicht viel Geld, aber wenn man bedenkt, dass man vielleicht mehrere Projekte gleichzeitig unterstützen will, ist das vielleicht zu viel, besonders da vielleicht nicht jedes Projekt so viel wert ist, vielleicht will man auch einfach nur jedem Projekt, das man nutzt, einen Dollar zukommen lassen oder Ähnliches. Kurz gesagt: Das Problem ist, dass noch kleinere Beträge im Moment nicht effektiv transportiert werden können, da die Gebühren viel zu hoch sind (Stichwort: Micropayment).

8.5 Verzögerte Releases

Eine weitere Möglichkeit, mit Freier Software Geld zu machen, besteht darin, die Software in den ersten ein oder zwei Jahren wie normale Software zu verkaufen und dann, wenn sie keinen Gewinn mehr abwirft, bzw. wenn eine neue Version rauskommt, die alte Software als Freie Software freizugeben. Somit hat derjenige, der warten kann, eine ordentliche und erprobte Freie Software und derjenige, der bereit ist Geld abzugeben, bekommt immer die neueste Version. Als Beispiel wäre hier Ghostscript³¹ anzuführen, die mit GNU Ghostscript eine sehr wichtige Komponente für ein GNU/Linux System liefern, nämlich sozusagen den Druckertreiber, bzw. eigentlich sogar noch deutlich mehr als das. Bei Ghostscript ist die aktuelle Version immer kostenlos für private Nutzung und kostet Geld für kommerzielle Nutzung³². Bei einer neuen Release wird dann jeweils die vorherige Version unter die GNU GPL gestellt. Als weiteres Beispiel wäre IDSoftware anzuführen, die ihre indizierten Spiele Doom und Quake mehrere Jahre nach dem Release, wenn sie kommerziell uninteressant geworden sind, ebenfalls unter die GNU GPL stellen. Als Einschränkung wäre hier aber zu erwähnen, dass nur der Quellcode unter der GNU GPL steht, die Level und Grafiken muss man sich weiterhin durch den Kauf des Original-Spiels erwerben.

Es gibt noch eine Sonderform von verzögerten Releases, nämlich Software, die mit Sourcecode ausgeliefert wird, aber bestimmte Freiheiten einschränkt, in der Regel die, die modifizierten Quellen weiterzuverteilen oder bestimmte Änderungen vorzunehmen. In der Lizenz ist dann festgehalten, dass die Software unter eine freie Lizenz fällt, sobald bestimmte Bedingungen nicht mehr erfüllt sind, meist etwa, wenn die Firma bankrott geht oder sie die Software

³¹ <http://www.ghostscript.com>

³² http://www.ghostscript.com/pages/aladdin_license.html

nicht mehr weiterentwickelt. Der Nachteil an solch einem Modell ist, dass es nicht klar ist, wann und ob in absehbarer Zeit die Software überhaupt zu Freier Software wird, problematischerweise würde sie es am ehesten werden, wenn man die Software nicht benutzt und so mehr oder minder den Bankrott der Firma herbeiführt. Ein Beispiel für ein solches Modell wäre z. B. Bitkeeper³³.

8.6 Fazit

Businessmodelle, um direkt Freie Software profitabel zu erstellen, gibt es wenige, bzw. nur dann, wenn die daraus resultierende Software nur für sehr spezialisierte Anwendungszwecke gedacht ist und dementsprechend der Nutzerkreis klein ist, am besten ist wohl „Softwareprodukt“:„Nutzer“ im Verhältnis 1:1, denn dann kann die Freie Software praktisch wie normale proprietäre Software verkauft werden. Freie Software funktioniert nun mal nicht wie normale proprietäre Software, wo man eine große Nutzerschaft dazu bringen kann ein einziges Produkt zu kaufen, da jeder, der das Produkt erwirbt, auch damit das Recht erwirbt es weiter zu verbreiten.

Es gibt allerdings diverse auch gut funktionierende Modelle, Dienstleistung um Freie Software herum aufzubauen, wie man z. B. bei SuSE oder Red Hat sehen kann. Dabei springen auch sehr viele, meist kleinere, freie Softwareprodukte heraus, die sind allerdings in der Regel nur ein Nebenprodukt und eben nicht etwas, womit eine Firma große Gewinne machen kann. Es besteht hier also ein meist, aber nicht zwangsläufig, ausgewogenes Verhältnis zwischen Geben und Nehmen aus dem Pool der Freien Software.

Letztendlich zielt das Modell der Freien Software ja auch darauf hin ab, dem Nutzer einer Software alle Möglichkeiten zu geben, die auch der Distributor der Software hat, ein Abhängigkeitsverhältnis, Upgradezwang und Ähnliches wird somit im Keim erstickt und macht Freie Lizenzen für kommerzielle Software nicht besonders interessant. (ir)

8.7 Vor- und Nachteile Freier Software

8.7.1 Innovationsarmut

Sehr viele Software im Bereich der Freier Software ist letztendlich darauf hinaus proprietäre Software nachzumachen, teilweise ist das Resultat daraus der proprietären Software überlegen, teilweise bleibt sie aber auch hinter dem proprietären Produkt zurück. Letztendlich hat die Vergangenheit aber gezeigt, dass Freie Software nicht zu Innovationen neigt. Innovative Ideen,

³³ <http://www.bitkeeper.com>

bzw. Implementationen derselben haben oft das Problem genug Leute zu finden, die daran mitarbeiten. Letztendlich ist es wohl deshalb so, weil es einfacher ist etwas Bekanntes zu duplizieren, als etwas von Grund auf Neues zu machen.

8.7.2 Usability

Viele Freie Software hat nicht ganz zu Unrecht den Ruf „von Programmierern für Programmierer“ zu sein. Usability-Tests, wie sie bei kommerzieller Software zu finden sind, gibt es bei Freier Software eher selten, bzw. wenn, dann werden sie eben von den Benutzern selbst gemacht und die sind meist schon recht „schmerzresistent“, wodurch grobe Schnitzer in der Usability gerne verziehen werden. Ansätze, dies zu ändern, gibt es aber reichlich, sowohl beim Gnome, als auch beim KDE-Projekt wird darauf geachtet und dementsprechend ist dort auch der meiste Fortschritt bei der Usability zu sehen. Inkonsistenzen zum Restsystem bleiben aber leider nicht aus, da KDE und Gnome auch nur ein Teil des Ganzen sind.

8.8 Maintenance

Allerdings hat Freie Software auch teilweise deutliche Vorteile gegenüber proprietärer Software, diese sind im wesentlichen durch die gegebene Freiheit bedingt. Als Beispiel wäre hier das Installieren von Treibern zu erwähnen, bei einem Proprietären System muss man sich seine Treiber und Updates derselben in aller Regel beim Hersteller der Hardware zusammensuchen, was vergleichsweise aufwändig werden kann, wenn man Hardware von verschiedenen Herstellern einsetzt. Bei Freier Software bekommt man alle Treiber in der Regel aus einer Quelle, meist dem Distributor, das heißt, man muss sich nicht die Treiber zusammensuchen, sondern sie werden bei der Distribution mitgeliefert. Ähnlich sieht es mit der Software selbst aus, Updates bei proprietärer Software müssen ebenso mühsam zusammengesucht oder gar gekauft werden, bei Freier Software bekommt man sie komplett und fertig zum Installieren beim Distributor. Updates lassen sich so teilweise mit Null Aufwand einspielen, was im Gegensatz zu proprietärer Software ein enormer Vorteil ist. Ganz so einfach ist es aber nur, wenn man seine Software direkt vom Distributor bezieht. Wenn man sie aus den Upstream-Quellen³⁴ beziehen will, ist die Installation und das Updaten oft deutlich komplizierter. Ein

³⁴ Upstream bezeichnet in der Regel den Ort, wo die Software ursprünglich hergekommen ist, der Upstream-Autor ist also z. B. derjenige, der die Software geschrieben hat. Distributoren liefern in der Regel nicht denselben Code wie der Upstream, sondern verändern diesen minimal, um ihn an die distributionsspezifischen Gegebenheiten anzupassen.

weiterer wesentlicher Vorteil bei Freier Software ist, dass man sie verteilen, kopieren und verändern kann, bei proprietärer Software ist das aufwändig und kostspielig (Lizenzen kaufen/verwalten) bzw. illegal.

8.9 Neue Technik

Bei neuen Techniken oder Dateiformaten, wie MP3, QuickTime oder MPEG4 hat Freie Software oft ein Problem. Dies liegt aber nicht, wie andere Probleme, in der Art wie Freie Software hergestellt wird begründet, sondern liegt im Wesentlichen an schlecht oder gar nicht dokumentierten Dateiformaten oder an Patentproblemen, die das Implementieren von freien Readern oder Writern für ein Dateiformat unmöglich machen, so gibt es zwar Encoder für MP3, allerdings sind diese aufgrund von Patenten in einigen Ländern illegal und können somit nicht benutzt werden. Bei QuickTime, bzw. genauer dem Sørensen-Codec, dem momentan wohl meistbenutzten Format, wenn es um Filme oder Trailer im Internet geht, ist das Problem noch schlimmer, hier ist sowohl das Enkodieren als auch das Dekodieren ohne lizenzierten Player untersagt, es existiert also keine Freie Software, um QuickTime-Filme abzuspielen. Letztendlich schaut es hier bei proprietärer Software nicht anders aus, alternative Player gibt es in der Regel nur sehr selten.

Die mangelnde Dokumentation von Hardware macht es Autoren Freier Software auch nicht einfach, Treiber für eine Hardware zu schreiben, die alle Features dieser benutzt. So ist es zwar oft so, dass ein Treiber existiert, dieser allerdings oft hinter einem proprietären Treiber hinterherhinkt. Bei gut dokumentierter Hardware kann man allerdings auch oft beobachten, dass die Hardware mit einem freien Treiber besser oder flexibler läuft, da diese in der Regel besser gepflegt werden als ein proprietärer. (ir)

9 Open Source und die Gesellschaft

Was hat sich tatsächlich geändert? Open Source/Freie Software rückte in den letzten Jahren immer mehr in den Blickpunkt der Öffentlichkeit. Vielleicht ist es für uns als Mitglieder dieser Community, die wir betrachten, überhaupt nicht möglich, eine Sicht „von außen“ zu liefern – das mag aber gerade unser Vorteil sein, können doch so die Hoffnungen, die in dieser Bewegung liegen, voll gezeigt werden. Wie stellen wir uns nun freie Software in der Öffentlichkeit vor?

Die doch sehr philosophisch begründete GPL zeigte bereits, wie mit der geschickten „Ausnutzung“ der geltenden Gesetze, d.h. des Prinzip des geistigen Eigentums, des unaufhebbaren Copyrights, eben jenes ins Gegenteil, in

das sogenannte Copyleft, verkehrt werden kann. Das Recht des Autors, über sein Werk verfügen zu können und seine geistige Schöpfung (wobei dieser Begriff sicherlich diskussionswürdig ist) beliebig ausschachten zu können, wird genutzt, um genau dieses nicht zu tun, die Informationen werden „befreit“. Genau wie ein gesprochenes Wort nicht zurück genommen werden kann, ist der Code, sobald er dem Copyleft unterliegt, nicht mehr aus dem öffentlichen Wissensschatz zurücknehmbar³⁵.

Die Idee, dass Informationen zwar einen Wert haben, aber prinzipiell für jedermann zugänglich sein sollten, ist nicht neu. Insbesondere in der Wissenschaft geht es gar nicht anders (s. Abschnitt 9.1). Neu hingegen ist die Hoffnung, dass sämtliche Information, die für andere einen Nutzen haben mag, ganz selbstverständlich frei wird. Das WWW hat geholfen, einen Teil dieser Vision wahr werden zu lassen: Schaut man sich eine HTML-Seite an, habt man automatisch auch Zugang zum Quellcode, die Information wird nicht in einer Art kompilierter Form geliefert, sondern der Quellcode selbst ist die Information. Jeder, der vor hat, eine Webseite zu gestalten, kann sich den Quellcode einer Seite, die er für besonders gut gelungen hält, ansehen und die verwendeten Ideen für sich nutzbar machen. Auch wenn der Autor sein Recht genutzt hat und durch eine entsprechende Zeile klargestellt hat, dass er die Weiterverwendung des HTML-Codes nicht wünscht, so gestattet er dennoch die Einsichtnahme in seine Ideen. Die Webautoren, die durch (leicht umgehbare) Skript-Tricks das Betrachten des Quelltexts verhindern wollen, haben das Prinzip nicht verstanden: Der Status quo von Informationen im Web ist frei, erst wenn der Autor sich bewusst dafür entscheidend, werden die Informationen unfrei.

Klargestellt werden sollte an dieser Stelle, worum es bei dem Begriff Information in diesem Zusammenhang überhaupt geht: Warum ist Freeware weniger frei als freie Software, ist sie doch kostenlos und jeder darf sie benutzen? Nehmen wir an, die Software sei für die Buchhaltung gedacht und noch nicht Euro-tauglich, habe ich dann die Freiheit, alles mit dieser sogenannten „Freeware“ zu tun, was ich will, nämlich das Programm um die Euro-Funktionen zu erweitern? Nein, natürlich nicht, weil eben die Sourcen nicht mitgeliefert werden. Damit ein Produkt wirklich frei ist, muss seine Bauanleitung dabei oder problemlos erhältlich sein. Natürlich ist es (noch) Utopie zu hoffen, dass dies für alle Produkte passiert – im Moment ist nicht daran zu denken, dass man für jeden Pullover, den man trägt, das Strickmuster erhalten kann oder jeder Kuchen aus dem Supermarkt gleich mit dem

³⁵ Einige Ausnahmen, in denen GPL-Code vom Autor in eine andere – unfreie – Lizenz überführt wurde, bestätigen hier die Regel. Sie haben allerdings mehr mit den Beschränkungen, denen das Copyleft im aktuellen gesetzlichen Rahmen unterworfen ist, zu tun, als mit der Philosophie, die es darstellt.

Rezept zu seiner Herstellung kommt. Darum geht es auch gar nicht. Vielmehr hoffen wir, dass sich wenigstens im Bereich der Software und Information ein Paradigmenwechsel ergibt, der dazu führt, dass Autoren ihre Quellen ganz selbstverständlich zum eigentlichen Produkt beilegen, weil es die Norm in der Gesellschaft geworden ist.

9.1 Wissenschaft

In den (Natur-)Wissenschaften ist schon seit jeher das „Open Source“-Prinzip die alleinige Methode, Ergebnisse zu publizieren. Das Schlagwort heißt hier *Peer-Review*: Damit Forschungsergebnisse allgemein anerkannt werden, müssen sie auch von anderen Wissenschaftlern nachvollziehbar sein. Wenn ein Wissenschaftler einen Artikel veröffentlichen sollte, der behauptet, dass er eine funktionierende Zeitmaschine gebaut hätte, müsste er eine detaillierte Beschreibung der zugrundeliegenden Theorie, Bauteilelisten für die Maschine und eine exakte Beschreibung, wie die Maschine funktioniert, mitliefern – ganz einfach aus dem Grund, dass ihm kein anderer Wissenschaftler glauben wird, bis sie nicht selbst eine derartige Maschine nachgebaut und erfolgreich eingesetzt hätten. Das gilt natürlich nicht nur für eine – zugegebenermaßen – recht revolutionäre Entdeckung wie die angesprochene Maschine, sondern für jede wissenschaftliche Entdeckung/Theorie.

So ist es durchaus im Interesse des Wissenschaftlers, sämtliche „Quellen“ mitzuliefern, nur dadurch kann er schließlich sein Ziel der Anerkennung und Glaubwürdigkeit erreichen. (mm)

10 Zukunft

Wie sieht es mit der Zukunft von Freier Software aus? Wird sie überleben, wird sie die Welt revolutionieren oder wird sie einfach nur ein Nischendasein fristen?

Als erstes wäre anzumerken, dass Freie Software kein Werbe-Gag oder Ähnliches einer grossen Firma ist, der verschwindet, sobald selbige keine Zeit mehr dafür hat. Freie Software ist im Wesentlichen „*Community driven*“, das heißt, Leute schreiben die Software, die sie später auch tatsächlich benutzen wollen, verwalten sie selbst und verteilen sie selbst. Es gibt zwar auch größere Firmen, die Freie Software unterstützen, allerdings leisten diese nur einen kleinen Beitrag, der größte Anteil kommt aus der Community.

Da Freie Software nicht zentral verwaltet oder erstellt wird, kann sie auch nicht an einem Punkt angegriffen werden. Sie ist also gegen singuläre Ereignisse relativ immun, wenn ein Entwickler ausfällt, findet sich meist in kurzer

Zeit ein neuer. Da Freie Software dem Nutzer die Freiheit gibt sie zu verändern, zu modifizieren und zu verteilen besteht auch keine Abhängigkeit zwischen Kunde und Produzent. Wenn dem Kunden etwas nicht gefällt, was der Produzent der Software macht, steht es dem Kunden frei, die Software zu nehmen und jemanden anders für die Weiterentwicklung zu bezahlen. Freie Software kann weder aufgekauft werden, noch kann die Freiheit, die sie bietet, nachträglich genommen werden. Software, die einmal frei war, bleibt für immer frei.

Gibt es dennoch Gefahren für Freie Software? Die gibt es in der Tat, das sind in der Regel Gesetzesänderungen oder Ähnliches, die nicht nur einen oder ein paar Dutzend Entwickler betreffen, sondern gleich einen Großteil. Da Freie Software zwar auch international eingesetzt und entwickelt wird, ist auch diese Gefahr gedämpft, da sich häufig ein Land mit anderen Gesetzen finden lassen würde, doch die Gefahr ist immer noch sehr groß, da die Behinderung enorm wären. Als Beispiel sei hier der DMCA (*Digital Millenium Copyright Act*) der USA angesprochen, der es verbietet Kopierschutzmaßnahmen und Ähnliches zu umgehen. Das schließt nun aber leider auch zum Beispiel die Verschlüsselung von DVDs ein, die im Wesentlichen kein Kopierschutz ist, sondern dafür sorgt, dass die DVD nur auf lizenzierten Playern, die den Region Code und Ähnliches beachten, abgespielt werden kann. So ist es mit Freier Software nicht möglich verschlüsselte DVDs anzuschauen, da die Entschlüsselungs-Software, die dafür benötigt wird, bekannt unter dem Namen *DeCSS*, unter den DMCA fällt und damit verboten ist.

10.1 Informationsgesellschaft

Im Hinblick auf die so oft postulierte Informationsgesellschaft bliebe es noch an der Reihe die Rolle der Freien Software hier zu betrachten. Digitale Information ist letztendlich nur eine bestimmte Reihenfolge von Bits, sie lässt sich verlustfrei kopieren, verteilen und betrachten, ohne dass man dabei irgendwie eingeschränkt ist, wie das bei materiellen Dingen der Fall war. Ob man eine Kopie einer Information macht oder tausende, macht keinen großen Unterschied, ganz im Gegensatz zu herkömmlichen materiellen Dingen, wie einem Hammer oder einem Buch³⁶ wo kein Kopieren ohne erheblichen Aufwand möglich ist.

Das Problem, das dabei nun zweifellos auftritt, ist, dass die Information an sich erstmal frei ist, was natürlich ganz gegen das Interesse des Produzenten dieser Information ist, dieser will selbige in der Regel kontrollieren

³⁶ Letztendlich enthält ein Buch natürlich auch nur Informationen, aber es gibt keine direkte Möglichkeit diese Information einfach auszulesen oder zu vervielfältigen.

können, um letztendlich dadurch Geld zu verdienen. Allerdings kann der Produzent erstmal auch nichts Großartiges dagegen unternehmen. Stattdessen müssen Copyright und Patente, die ursprünglich gar nicht dafür gedacht waren, nun dafür herhalten, die Freie Information zu bändigen und deren Verteilen zu illegalisieren. Freie Software ist letztendlich nur der Versuch, durch Lizenzen wie die GNU GPL diese Urfreiheit von Information auch rechtlich zu festigen, zielt also genau in die andere Richtung. Wie wird sich das alles in Zukunft entwickeln? Und können Copyright und Patente einen Schutz gegen „Missbrauch“ von Informationen darstellen? Kurz gesagt, nein. Information ist frei und wenn ich eine Information anschauen kann, dann kann ich sie auch kopieren und, wenn nötig, digitalisieren und wenn sie erstmal digitalisiert ist, kann man ihre Verbreitung nur noch schwer kontrollieren. Besonders mit dem Aufkommen von Peer-to-Peer-Netzwerken wie Napster und Gnutella zeigt sich auch, dass die Kundschaft für „Information“ recht groß ist, bzw. dass auch die Anzahl derjenigen, die Informationen zur Verfügung stellen, recht groß ist. Durch das Copyright kann man dem Ganzen allerdings noch vergleichsweise einfach ein Ende machen, zumindestens dann, wenn ein paar Gesetze zum Schutz der Privatsphäre gelockert werden. Denn aktuelle Peer-to-Peer-Netzwerke sind nicht anonym, das heißt spätestens der Internet Provider kann an Hand der IP herausbekommen, wer welche Inhalte runterlädt oder verteilt. Wenn also den „Produzenten“ durch entsprechende Gesetze ein Einblick auf die Daten der Provider gegeben würde, wäre es mit dem heutigen Peer-to-Peer schnell zu Ende.

Allerdings ist die nächste Generation von Peer-to-Peer-Netzwerken schon in Arbeit, diese läuft dann verschlüsselt und größtenteils anonymisiert ab. Peek-A-Booty³⁷ und Freenet³⁸ sind hier wohl die bekanntesten. Mit solchen anonymisierten Netzwerken besteht dann selbst mit Einblick in die Providerdaten praktisch keine Möglichkeit mehr rauszufinden, wer welche Inhalte zur Verfügung stellt oder runterlädt. Man könnte so etwas letztendlich nur noch durch die Illegalisierung von Verschlüsselung aushebeln und dann entsprechend präventiv Bestrafung machen, wenn Verschlüsselung benutzt wird. Der schon erwähnte DMCA geht schon in etwa in diese Richtung. Sachen wie ein Zwangs-Digital-Rights-Management (DRM) könnten ebenfalls gegen den freien Informationsaustausch vorgehen, wobei etwas wie DRM nur funktionieren kann, wenn es in *jedes* Gerät, das bei der Datenübertragung eine Rolle spielt, eingebaut ist. Das heißt also, dass so etwas nicht funktionieren wird, da man dazu erstmal überall neue Hardware bräuchte und vor allem dafür sorgen müsste, dass nur noch DRM-fähige Hardware auf den Markt kommt,

³⁷ <http://www.peek-a-booty.org/>

³⁸ <http://freenetproject.org/>

was ohne Gesetzesänderung nicht möglich sein wird. Außerdem würde eine solche praktisch zur Illegalisierung sämtlicher heutiger Hard- und Software führen, was nicht sehr realistisch scheint. Letztendlich kann alle Verschlüsselung beim Copyright-Schutz nur wenig helfen, denn spätestens zu dem Zeitpunkt, an dem der Kunde die Information betrachtet, liegt sie unverschlüsselt vor und kann abgegriffen und kopiert werden.

Letztendlich sind Informationsgesellschaft und Urheberrecht also zwei Dinge, die nicht so recht zusammenpassen und in Zukunft wird sich das auch nicht ändern, da Information an sich frei ist. Freie Software sorgt letztendlich zu einem Teil dafür, dass das auch so bleibt, denn in Freier Software ist es praktisch unmöglich Kopierschutzvorrichtungen oder Ähnliches unterzubringen, denn sie ließen sich trivial aushebeln.

(ir)

Literatur

- [1] Jürgen Schmidt, *Linux versenkt die Titanic*,
<http://www.heise.de/newsticker/data/ju-07.01.98-000/>
- [2] Richard Sietman, *Wettbewerb im Gerichtssaal*, c't 17/2001, S. 170-181
- [3] Franz-Josef Schöniger, *Schlechte Karten für deutsche Entwickler?*,
c't 16/1999, S. 72ff
- [4] Richard Sietmann, *Software-Patente: Gegner zahlenmäßig überlegen*,
<http://www.heise.de/newsticker/data/jk-06.08.01-002/>
- [5] Dr. Oliver Diedrich, *Europäische Software-Patente durch die Hintertür*,
<http://www.heise.de/newsticker/data/odi-07.11.01-000/>
- [6] Nathan Cochrane, *Melbourne man patents the wheel*,
<http://www.theage.com.au/news/state/2001/07/02/FFX0ADFPLOC.html>
- [7] Institut für Rechtsfragen der freien und Open Source Software, *Open
RTLinux Patent License*, <http://www.ifross.de/>