Universiteit
Leiden
The Netherlands

Towards an improved regulatory framework of free software : protecting user freedoms in a world of software communities and eGoverments

Siewicz, K.

Citation
Siewicz, K. (2010, April 20). *Towards an improved regulatory framework of free software : protecting user freedoms in a world of software communities and eGoverments*. *Meijers-reeks*. E.M. Meijers Institute of Legal Studies, Leiden. Retrieved from https://hdl.handle.net/1887/15276

Version:	Not Applicable (or Unknown)
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/15276

Note: To cite this publication please use the final published version (if applicable).

Towards an Improved Regulatory Framework of Free Software

Protecting user freedoms in a world of software communities and eGovernments

Krzysztof Siewicz

Towards_an_Improved_def.indd 1 23-02-2010 10:49:09

Towards_an_Improved_def.indd 2 23-02-2010 10:49:10

Towards an Improved Regulatory Framework of Free Software

PROTECTING USER FREEDOMS IN A WORLD OF SOFTWARE COMMUNITIES AND eGOVERNMENTS

PROEFSCHRIFT

Ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus prof. mr. P.F. van der Heijden,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 20 april 2010
klokke 15.00 uur

door

Krzysztof Siewicz

Geboren te Warschau, Polen in 1979

Towards_an_Improved_def.indd 3 23-02-2010 10:49:10

Promotiecommissie:

Promotores:	Prof. dr. H. J. van den Herik Prof. mr. A. H. J. Schmidt
Overige leden:	Prof. dr. F.M.T. Brazier (Delft University of Technology) Prof. mr. R.E. van Esch Prof. dr. H. Franken Prof. mr. drs. C. Stuurman (Universiteit van Tilburg) Prof. dr. D. Visser

SIKS dissertation series no. 2010-08
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research was partially funded by the Netherlands organization for international cooperation in higher education (Nuffic), as a part of the Huygens Scholarship Programme. The remaining part of this research was self-funded.

Lay-out: AlphaZet prepress, Waddinxveen

ISBN 978-83-930580-0-6

Copyright © Krzysztof Siewicz, 2010, this work is available under a Creative Commons Attribution-No Derivative Works 3.0 Unported License, full text of the license available at: http://creativecommons.org/licenses/by-nd/3.0/legalcode

Preface

The first microcomputer in my home came in the 1980s. Using it was not easy at all, certainly not when compared to using personal computers nowadays. At that time, it was necessary to know the commands (the source code) to let the computer do anything apart from blinking the command prompt. However, this was sufficiently intriguing to stimulate me to learn the basics of programming and attend a computer course at high school. During the course, around 1997, I first came across Free Software. It was a rough GNU/Linux distribution assembled by my high school teacher. Well, I must admit, it did not catch my attention then. The reason might have been that I already knew that I would like to study a different type of source code – the law. Paradoxically, becoming a lawyer has raised my interest in Free Software. Surely, source codes are important and knowing them is more than fun. But what matters even more are rights and obligations that come together with the code. Indeed, during my law studies I realised that the freedoms, which Stallman longs for, are legal rights. As a lawyer, I also understand that these freedoms cannot be taken for granted. Actually, it is not so easy to obtain the freedoms, and it is even harder to protect them. In other words, user freedoms tend to be very much alike other freedoms that the society struggled for in the past, and particularly in the recent past.

This observation has led me to setting myself a task: to design a framework that is able to protect the freedoms adequately. Then, I decided to make my task harder in two ways. First, I thought it would be a good idea to learn how the law actually works. So, while doing my research I have also practised law. I hope that this has strengthened the thesis in the direction of a more practice-oriented dissertation. Second, I submitted my ideas to Professor Jaap van den Herik and Professor Aernout Schmidt and asked them to become my supervisors. They willingly accepted and as a consequence my task became a crash-course on proper research, reasoning, and writing. They revised each chapter of this thesis some hundreds of times. Perhaps the most important thing that Jaap has taught me is to do everything to make the reader feel comfortable. Aernout has taught me in particular not to expect that problems will solve themselves. Having them both as supervisors was a thrilling experience, which I enjoyed even more as the final result approached.

When I now give my work a second thought, in retrospect the task was much more easy. I have had the firm support of my parents and my beloved wife. Whenever I came to Leiden, my Dutch friends were eager to help me. In particular, I wish to thank Franke van der Klaauw, Hugo Kielman, and Wouter Koelewijn who made me feel at home. I also want to thank for the free (as in "freedom") atmosphere that the partners and my colleagues at Grynhoff Woźny Maliński have created.

Warszawa, August 2009

Towards_an_Improved_def.indd 5 23-02-2010 10:49:10

Towards_an_Improved_def.indd 6 23-02-2010 10:49:11

Table of Contents

Preface 5

List of Abbreviations 11

List of Legislations 13

List of Model Licenses 15

List of Cases and Patents 17

List of Figures 19

List of Tables 19

1 Introduction
- 1.1 The main scene of play 22
  - 1.1.1 The software scene 22
  - 1.1.2 The standards scene 24
  - 1.1.3 Four combinations of software and standards 26
- 1.2 Actors and their audience 27
  - 1.2.1 Actors and their audience in the software scene 27
  - 1.2.2 Actors and their audience in the standards scene 31
- 1.3 Problem statement 33
- 1.4 Research questions 40
- 1.5 Research methodology 40
- 1.6 Structure of the thesis 42

2 Definitions and basic notions 43
- 2.1 Free Software 43
  - 2.1.1 Subject matter of the Free Software Definition 45
  - 2.1.2 Requirements of the Free Software Definition 45
  - 2.1.3 Addressees of the FSD 49
  - 2.1.4 Conclusion on Free Software Definition 51
- 2.2 Open Source Software 52
- 2.3 Open standards 53
- 2.4 Software communities 56
- 2.5 eGovernment 57

3 Regulatory framework of Free Software 59
3.1 Identification of rules for software and relations between them 62
3.1.1 Free Software licenses 63
3.1.2 Other rules for software 73
3.1.3 Conclusion on rules for software and relations between them 95
3.2 Identification of rules for standards and relations between them 96
3.2.1 Rules that lead to closed standards 96
3.2.2 Rules that lead to open standards 103
3.2.3 Conclusion on rules for standards and relations between them 110
3.3 Reconstruction of a model of the current framework 111
3.3.1 Rules that regulate software 111
3.3.2 Rules that regulate standards 111
3.3.3 Regulatory environment 112
3.3.4 Graphical presentation of the model of the current framework 114
3.4 Chapter conclusions 116

4 Communitarian protection of user freedoms 117
4.1 Limitations and restrictions of the freedoms 118
4.1.1 License proliferation and incompatibilities 120
4.1.2 License revocability 122
4.1.3 *Inter partes* nature of licenses 123
4.1.4 Software-related patents 125
4.1.5 Contracts with distributors 126
4.1.6 Liability rules 127
4.1.7 Non-legal regulators of software 128
4.1.8 Closed standards 128
4.1.9 Regulatory environment 129
4.1.10 Conclusion on limitations and restrictions of the freedoms 129
4.2 The freedoms in software communities 130
4.2.1 License proliferation and incompatibilities 134
4.2.2 License revocability 138
4.2.3 *Inter partes* nature of licenses 139
4.2.4 Software-related patents 141
4.2.5 Contracts with distributors 142
4.2.6 Liability rules 143
4.2.7 Non-legal regulators 144
4.2.8 Closed standards 145
4.2.9 Regulatory environment 145
4.2.10 Conclusion on the freedoms in software communities 146
4.3 Chapter conclusions 147

5 User freedoms in eGovernments 149
5.1 User freedoms in a world without eGovernments 153
5.2 User freedoms in Closed eGovernments 154
5.2.1 Closed eGovernment (A) 155
5.2.2 Closed eGovernment (B) 156
5.2.3 Semi-Closed eGovernment 157
5.2.4 Evaluation of user freedoms in Closed eGovernments 159
5.3 User freedoms in Open eGovernments 160
5.3.1 Open eGovernment (A) 161
5.3.2 Open eGovernment (B) 165
5.3.3 Supra-Open eGovernment 167
5.3.4 Evaluation of user freedoms in Open eGovernments 172
5.4 Chapter conclusions 172

6 Proposal of an improved regulatory framework 175
6.1 Inefficiencies of the current framework and possible improvements 176
6.1.1 License proliferation and incompatibilities 177
6.1.2 License revocability 179
6.1.3 *Inter partes* nature of licenses 181
6.1.4 Software-related patents 182
6.1.5 Contracts with distributors 184
6.1.6 Liability rules 185
6.1.7 Non-legal regulators of software 187
6.1.8 Closed standards 188
6.1.9 Regulatory environment 191
6.1.10 Evaluation of the inefficiencies of the current framework 192
6.2 Appropriate improvements in the framework 193
6.2.1 Proper organization of communities 193
6.2.2 Regulation of eGovernments 196
6.2.3 Free Software legislation 198
6.2.4 Restriction of software-related patents 199
6.2.5 Promotion of open standards 202
6.2.6 Internationalization of the framework 204
6.3 Construction of a proposal of an improved framework 206
6.4 Chapter summary 207

7 Conclusions and further research 211
7.1 Answers to research questions 211
7.2 Answers to the problem statement 214
7.3 Provisional impact of our proposed framework 215
7.3.1 Rules created if the improvements are implemented 215
7.3.2 Impact of the new rules on the existing rules 218
7.3.3 Conclusions on the provisional impact of our proposed framework 220
7.4 Further research 221

References 223

Summary 229

Samenvatting 231

Curriculum vitae 234

Siks Dissertation Series 235

Meijers Ph.D. list 242

Towards_an_Improved_def.indd 10 23-02-2010 10:49:11

List of Abbreviations

API	Application Programming Interface
Art.	article
BSD	Berkeley Software Distribution
CeCILL	Ce(A)C(nrs)I(NRIA)L(ogiciel)L(ibre)
cf	confer, (*lat.*) "compare"
CFI	Court of First Instance (of the European Communities)
Dz. U.	Dziennik Ustaw, Polish official journal for the publication of laws
ECJ	European Court of Justice
ed.	editor(s)
e.g.	exempli gratia, (*lat.*) "for example"
EPO	European Patent Office
et al.	et alii, (*lat.*) "and others"
etc.	et cetera, (*lat.*) "and so on"
et seq.	et sequens, (*lat.*) "and the following"
ETSI	European Telecommunications Standards Institute
EIF	European Interoperability Framework
FN	footnote
FSD	Free Software Definition
FSF	Free Software Foundation
GNU	GNU is Not Unix (a recursive acronym)
HTML	Hypertext Mark-up Language
htm, html	a suffix of a name of a file written in HTML
http://	hypertext transfer protocol
id.	idem, (*lat.*) "the same"
IDABC	Interoperable Delivery of European Government Services to public Administrations, Businesses and Citizens, a programme managed by the European Commission

i.e.	id est, (*lat.*) "that is"
IEC	International Electrotechnical Commission
ISO	International Standards Organization
IT	Information Technology
NDA	non-disclosure agreement
NSA	Naczelny Sąd Administracyjny, Polish Supreme Administrative Court
OJ	Official Journal of the European Community
op. cit.	opus citatum, (*lat.*) "the work cited"
OSD	Open Source Definition
OSI	Open Source Initiative
PDF	Portable Document Format
pdf	a suffix of a name of a file in a PDF
PKN	Polski Komitet Normalizacyjny, Polish Standardization Committee (Polish national SSO)
RAND	reasonable and non-discriminatory
RF	royalty-free
Sec.	section
SSO	standard-setting organization
U.S.	The United States of America
WIPO	World Intellectual Property Organization
WTO	World Trade Organization
WWW	The World Wide Web
ZUS	Zakład Ubezpieczeń Społecznych, Polish Social Insurance Office

List of Legislations

Berne Convention	Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979)
Copyright Treaty	WIPO Copyright Treaty (1996)
DMCA	Digital Millennium Copyright Act, 112 Stat. 2860 (1998)
EC Treaty	Treaty Establishing the European Community (as amended on 2 October 1997 by the Amsterdam Treaty)
Electronic Markets Competition Directive	Directive 2002/77/EC on competition in the markets for electronic communications networks and services (OJ L 249, 17.09.2002, P. 21)
EPC	European Patent Convention (1973)
E-Commerce Directive	Directive 2001/31/EC on certain legal aspects of information society services, in particular electronic commerce, in the Internal Market (OJ L 178, 17.07.2000, p. 1-16)
IT and Telecommunications Standardization Decision	Decision 87/95 of 22.12.1986 on standardization in the field of information technology and telecommunications (OJ L 36, 7.2.1987, P.31).
New York Convention	New York Convention on the Recognition and Enforcement of Foreign Arbitral Awards (1958)
Paris Convention	Paris Convention for the Protection of Industrial Property (as amended on September 28, 1979)
PCT	Patent Cooperation Treaty (as amended on October 3, 2001)
Polish Civil Code	Act of 23 April 1964 the Civil Code (Dz.U. of 1964 no. 16 item 93, as amended)
Polish Constitution	Act of 2 April 1997 the Constitution of the Republic of Poland (Dz.U. of 1997 no. 78 item 483)
Polish Copyright Act	Act of 4 February 1994 on copyright and neighbouring rights (Dz.U. of 2006 no. 90 item 631, consolidated version, as amended)

Towards_an_Improved_def.indd 13 23-02-2010 10:49:11

Polish Criminal Code	Act of 6 June 1997 the Criminal Code (Dz.U. of 1997 no. 88 item 553, as amended)
Polish Industrial Property Act	Act of 30 June 2000 Industrial Property Law (Dz.U. of 2003 no. 119 item 1117, consolidated version, as amended)
Polish Informatization Act	Act of 17 February 2005 on the informatization of entities performing public tasks (Dz.U. of 2005 no. 64 item 565, as amended)
Polish Public Information Act	Act of 6 September 2001 on access to public information (Dz.U. of 2001, no. 112 item 1198, as amended)
Polish Unfair Competition Act	Act of 16 April 1993 on unfair competition (Dz.U. of 2003 no. 153 item 1503 consolidated text, as amended)
Public Procurement Directive	Directive 2004/18/EC on the coordination of procedures for the award of public works contracts, public supply contracts and public service contracts (OJ L 134, 30.4.2004, p. 114.)
Public Sector Information Directive	Directive 2003/98/EC on the re-use of public sector information (OJ L 345, 31.12.2003, p. 90.)
Rental Directive	Directive 92/100/EEC on rental right and lending right and on certain rights related to copyright in the field of intellectual property (OJ L 346, 27.11.1992, p. 61-66)
Software Directive	Directive 91/250/EEC on the legal protection of computer programs (OJ L 122, 17.5.1991, p. 42–46)
Standards Directive	Directive 98/34/EC laying down a procedure for the provision of information in the field of technical standards and regulations and of rules in information society services (OJ L 204, 21.7.1998, P 37.)
TRIPS	WTO Agreement on Trade-Related Aspects of Intellectual Property Rights (1994)
UCC	Uniform Commercial Code
UCITA	Uniform Computer Information Transactions Act
U.S. Copyright Act	Copyright Act, 17 U.S.C. Sec. 101 (1976, as amended)

List of Model Licenses

AGPL	Affero General Public License, at: http://www.fsf.org/licensing/licenses/agpl.html
BSD-type license	a type of a permissive Free Software license, originally used for the Berkeley Software Distribution
CeCILL	a set of model licenses drafted within the Ce(A)C(nrs) I(NRIA)L(ogiciel)L(ibre) project, available at: http://www.cecill.info/licences.en.html
Eclipse Public License	a model license available at: http://www.eclipse.org/org/documents/epl-v10.php
EUPL	European Union Public License, at: http://europa.eu.int/idabc/en/document/2623/5585#eupl
FLA	Fiduciary License Agreement, at: http://www.fsfeurope.org/projects/ftf/FLA.en.pdf
GPL, GPLv2, GPLv3	GNU General Public License, its version 2 or 3, respectively, at: http://www.fsf.org/licensing/licenses/gpl.html
LGPL	GNU Lesser General Public License, at: http://www.fsf.org/licensing/licenses/lgpl.html
MPL	Mozilla Public License, at: http://www.mozilla.org/MPL/MPL-1.1.html
MSPL	Microsoft Public License, at: http://opensource.org/licenses/ms-pl.html
Open Software License	a model license available at: http://www.opensource.org/licenses/osl-3.0.php.
SleepyCat License	a model license available at: http://www.opensource.org/licenses/sleepycat.php
Sun Industry Standards Source License	a model license available at: http://www.OpenOffice.org/licenses/sissl_license.html

Towards_an_Improved_def.indd 16 23-02-2010 10:49:12

List of Cases and Patents

European cases

13/77, *SA G.B.-Inno-B.M. v. ATAB*, 16.11.1977, 1977 E.C.R. 02115
C-18/88, *RTT v. GB-Inno-BM SA*, 11.01.1988, 1991 E.C.R. I-05941
66/86, „*Ahmed Saeed Case*", 11.04.1989, 1989 E.C.R. 00803
C-35/96, *Commission v. Italy*, 18.06.1998, 1998 E.C.R. I-03851
C-198/01, „*CIF Case*", 9.9.2003, 2003 OJ C 264 P. 9
Case T-201/04, *Microsoft v. the Commission*

U.S. cases

Aro Mfg. Co. v. Convertible Top Replacement Co. 377 U.S. 476, 84 S.Ct. 1526 (U.S.Mass. 1964.)
In re Bilski 545 F.3d 943 (Fed. Cir. 2008) (en banc)
Jacobsen v. Katzer 535 F.3d 1373 (2008).
Lotus Development Corporation v. Borland International, Inc. 516 U.S. 233 (1996)).
M.A. Mortenson Co. v. Timberline Software Corp. 970 P.2d 803 (Wash. Ct. App. 1999), aff'd, 998 P.2d 305 (Wash. 2000)
Microstar v. Formgen, Inc. 942 F. Supp. 1312 (S.D. Cal. 1996), aff'd in part, rev'd in part on other grounds, 154 F.3d 1107 (9th Cir. 1998)
ProCD v. Zeidenberg 86 F.3d 1447 (7th Cir. 1996)
Rambus Inc. v. Infineon Technologies Holding North America Inc. 318 F.3d 1081 (C.A.Fed. Va. 2003)
State Street Bank & Trust v. Signature Financial Services 149 F.3d 1368 (Fed. Cir. 1998), cert. denied, 119 S.Ct. 851 (U.S. Jan 11, 1999)
Symbol Technologies Inc. v. Proxim Inc. 2004 WL 17701290 (D.Del.)
Wallace v. Free Software Foundation, at: http://www.groklaw.net/pdf/WallaceFSFGrantingDismiss.pdf.

German cases

Harald Welte v. D. GmbH, at: http://www.jbb.de/judgment_dc_frankfurt_gpl.pdf.

Polish cases

decision of the Polish Supreme Court of 20 May 1999 (I CKN 1139/97)
decision of the Polish Supreme Court of 28 November 2003 (IV CK 206/2002)
decision of the Polish Supreme Administraive Court dated 16 September 2004 (OSK 600/04)

Towards_an_Improved_def.indd 17 23-02-2010 10:49:12

European Patent Convention cases

International Business Machines, Corp./Computer program product, Decision of the EPO Technical Board of Appeal 3.5.1 dated 1 July 1998, T 1173/97 (OJ 10/1999, 609)

Patents

PL 123 820, published on 25 September 1984
PL 116 724, published on 31 March 1983

Towards_an_Improved_def.indd 18 23-02-2010 10:49:12

List of Figures

Figure 1.1: Actors and audience in proprietary software 28
Figure 1.2: Actors and audience in Free Software 30
Figure 2.1: Relations between the FSD and copyright law 47
Figure 2.2: Addressees of the FSD 50
Figure 3.1: Example development and distribution chains 60
Figure 3.2: Proliferation of licenses 74
Figure 3.3: Using programs under different Free Software licenses 75
Figure 3.4: Incompatibility of licenses 78
Figure 3.5: Rules for software and relations between them 95
Figure 3.6: Rules for standards and relations between them 110
Figure 3.7: Uniformity 112
Figure 3.8: Regulatory environment 112
Figure 3.9: Graphical presentation of the model of the current framework 115
Figure 4.1: Copyleft 123
Figure 4.2: The right to fork 124
Figure 6.1: Our improved framework as proposed in Chapter 6 207

List of Tables

Table 1.1: Four combinations of software and standards 26
Table 3.1: Four combinations between who distributes and what is distributed 61

Towards_an_Improved_def.indd 19 23-02-2010 10:49:12

Towards_an_Improved_def.indd 20 23-02-2010 10:49:12

1 Introduction

We live in a world full of information technologies. These technologies support us in an increasing number of tasks. The diversity of tasks in which they are able to support us is dazzling. For example, currently almost complete virtual worlds are created, in which everybody may play a role of their own choice with an entertainment scene as background. Similar technologies as used there may provide researchers with powerful tools for scientific data gathering and analysis. Additionally, the technologies facilitate smooth and worldwide marketing, in such a way that entrepreneurs can easily meet demands of distant and diverse customers. Communications and electronic commerce pleasantly coincide, the technologies allow us to organize teamwork efficiently. At the same time, governments all over the world use the technologies in an attempt to organize the governance of society, by introducing various eGovernment schemes. eGovernment may be an aid to the execution of our legal rights and obligations. It may also aim at promoting the participation of the whole society in decision-making processes in matters of public interest and at facilitating the exchange of public information. In many different support technologies Free Software plays an important role.

This thesis is based on the premise that a worldwide regulatory framework should be adopted to protect users of Free Software. Such a framework should consist of rules (and relations between them) that protect user freedoms as articulated by Richard M. Stallman.[1] In our study, we analyse the protection of the users in the world of (1) software communities and (2) eGovernments, as both of these phenomena influence user freedoms. Therefore, we start analysing whether and to what extent the current regulatory framework protects the freedoms under such influence. The players and the issues they are dealing with are considered as the main scene of play. It is our task to describe their roles and the consequences of their behaviour. If our analysis leads to the conclusion that the protection is not adequate, we will aim at proposing an improved regulatory framework.

In this chapter we provide an introduction to the analysis of the current framework. In Section 1.1, we describe the main scene of play, that is the world where human beings and computer programs cooperate. In Section 1.2, we introduce the actors of this scene and their audience. In Section 1.3, we formulate our Problem Statement (PS). We do this by taking a closer look

1 R.M. Stallman, *Free Software Definition*, at: http://www.gnu.org/philosophy/free-sw.html. An alternative version of the same document is available at: http://www.fsf.org/licensing/essays/free-sw.html.

at software communities and eGovernments in relation to user freedoms. In Section 1.4, we take the regulatory framework and the user freedoms as articulated by Stallman to formulate three Research Questions (RQs). In Section 1.5, we present the research methodology used in the thesis in order to answer the RQs. In Section 1.6, we give the structure of the thesis.

1.1 The main scene of play

It is hard to imagine (1) what the possibilities will be of an increasing technological support, as well as (2) which limitations we will face. Certainly, the support comes in the form of intelligent computer programs. Some of the programs may be special or even unique and will thus perform specific tasks, but there will also be many tasks that may be accomplished by programs that are massively produced and do not have to be customized much. Some of these programs may be instructed to perform their tasks autonomously. Additionally, the programs may be designed to communicate with each other, or with our human counterparts over a network. So, we envisage a world where human beings and computer programs cooperate to accomplish any imaginable task that may be undertaken by a human alone or by a group of humans or by intelligent agents.[2]

A minimal requirement for a world of seamless cooperation between humans and computer programs is that all these programs work efficiently and properly. Below, we distinguish between two layers of such a world: (1) the layer related to *software* itself and (2) the layer related to *standards* used by software to interoperate. We would like to call these layers *scenes*, since it facilitates our description. The two scenes host a variety of activities that affect the working of the programs. We describe them in Subsection 1.1.1 and 1.1.2, respectively. In Subsection 1.1.3, we complete the description by presenting four combinations. They result from combining two extreme types of the software spectrum with two extreme types of the standards spectrum.

1.1.1 The software scene

It is well known that so far (1) every software contains errors ("bugs") and (2) no software possesses all features that users need. Hence, if programs are to work efficiently and properly, their development should be organized in a

2 In general, an agent can be a human or a computer. The notion "intelligent agents" originates from the domain of Artificial Intelligence. An intelligent agent is a collection of software that is able to take intelligent decisions, and to act intelligently. Most agents do so autonomously (for lawyers this is a difficult point to understand). In computer science, research aims at the development of BDI agents (*i.e.*, agents with a belief, a desire, and an intention), see, *e.g.*, Wikipedia, *BDI software agent*, at: http://en.wikipedia.org/wiki/BDI_software_agent.

way that minimizes the occurrences of bugs. Also, in the ideal case all necessary features should be included in programs before users start using them. However, many bugs are revealed only after software is already distributed to users. Similarly, only then it often becomes apparent that some features of the program are not in place. This means that appropriate repairs ("patches") should be developed while the program is already serving its supportive purpose. Additionally, appropriate procedures for the development of feature upgrades (or updates) of the already used programs should be provided for. Moreover, after a patch, an upgrade, or an update (jointly: an improvement) is developed, it should be swiftly distributed to users so that they can benefit from the improved support and not have to continue using software that works improperly.

The need of correcting bugs and including new features in a dynamic environment means that it is not possible to use a given program unmodified for a significantly long time. Usually, every now and then the program has to be improved to remove its bugs or add new features. Since improvements require modifications, virtually all software is under continuous modification. For the time being, modification of software requires access to human-readable source codes.[3] Conversely, to use a program in its supportive function, a machine-readable object code ("binary") has to be made available to the user. After modification, source codes are translated into binaries in a process called "compilation".[4] Binaries may be distributed and used separately from source codes. Theoretically, if source codes are not available, a sufficiently skilled person may use binaries to fix bugs or include new features (*e.g.*, by way of decompilation).[5] But in practice, the most preferable way to modify a program is to access its source codes.[6]

Here, we may conclude that access to source codes is the first necessary condition for the ability to control the working of a program. In passing, we note that additional access to binaries is not necessary, since they may be

3 In the future, it may be left to autonomous non-human agents (see, *e.g.*, Wikipedia, *Self-modifying code*, at: http://en.wikipedia.org/wiki/Self-modifying_code).

4 We refer here to source codes written in compiled languages. There are also other programming languages that allow to execute source code directly, without compilation (interpreted languages). Programs written in interpreted languages are sometimes deliberately obfuscated in order to prevent users from accessing source codes. For the purposes of this thesis such an obfuscated code in an interpreted language may be considered as a "binary" as well. For more details see: Wikipedia, *Programming language*, at: http://en.wikipedia.org/wiki/Programming_language.

5 Decompilation is a way of reverse engineering a program, which leads to a reconstruction of source codes of the decompiled program from its binaries. See: Wikipedia, *Decompilation*, at: http://en.wikipedia.org/wiki/Decompilation.

6 One reason why decompilation is usually not preferred is the law. For example, under Art. 6 of the Software Directive, decompilation of a program is only allowed if it is indispensable to obtain the information necessary to achieve the interoperability of an independently created computer program with the decompiled program, and provided some other strict conditions are met at the same time.

rather easily produced using source codes.[7] By looking at (the number or weight) of the conditions on the access to source codes, it is possible to distinguish many different types of software. For our study we would like to identify the following two extreme types of software.

(1) **Proprietary software.** Computer programs are developed in private (the process of eliminating bugs and including new features is subject to exclusive control). This means that access to source codes is restricted to certain persons only. Also, only certain persons are allowed to distribute proprietary software. Additionally, there are restrictions on the use of proprietary software.

(2) **Free Software.** The participation in the development of computer programs is not restricted (the process of eliminating bugs and including new features is not subject to exclusive control). This means that access to source codes is not restricted. Also, there are no restrictions as to who is allowed to distribute Free Software. Obviously, there are no restrictions on the use of Free Software.[8]

Between these two extreme types we observe a spectrum of possibilities, all of which we call the *software scene*. It would be impractical to attempt to construct a regulatory framework for all of these possibilities and to analyse such a framework successfully. Thus, for the sake of clarity, in this thesis we would like to downsize the spectrum of the types of software to the two extremes mentioned above.

1.1.2 The standards scene

Usually, a computer program is not used in a vacuum. Currently, most programs are used together with other programs. For example, they communicate over a network. A program works efficiently and properly in such a situation if only it is able to *interoperate* with other programs. Programs interoperate using protocols, interfaces, and data formats. For one program to interoperate with another program, the program has to support a protocol, an interface, or a data format that the other program also supports. In other words, both programs have to use a common *standard* to interoperate. To make a program use the common standard, it has to be developed accord-

7 Assuming that a user has access to compilers or interpreters, as well as sufficient engineering skills. Since average user usually does not have such access or skills, access to binaries can be considered as necessary for such users to control the working of programs to the same extent as the access to source codes.

8 We consider Free Software simultaneously as a name and as a concept that should be written with capital letters. We concur with the majority of publications and will capitalize the concept throughout the text. We do not capitalize proprietary software. We present a more elaborate definition of Free Software in Section 2.1.

ingly. Thus, *access to the program's source code* is the first necessary condition of interoperability, since without source codes it is not possible to develop programs. However, it is not a sufficient condition.

Apart from having access to the program's source codes, one also has to know what is the common standard exactly. This information (usually referred to as "interoperability information") should most preferably be provided in a way accessible and understandable for a human being, in a document called "standard specification". If the information is not provided in such a way, it might still be possible for a sufficiently skilled person to *reverse engineer* another program that uses the standard in order to reconstruct the interoperability information. Reverse engineering can be performed in particular by (1) analysing the output of another program that already supports the standard, (2) decompiling the binaries of such a program, or (3) studying its source codes. However, reverse engineering is usually impracticable. The preferred method is to use the standard specification. Thus, the second necessary condition of interoperability is *access to the standard specification*.

Here, we may conclude that there are two necessary conditions for a program to work efficiently and properly with other programs: (1) access to the source codes of the program, and (2) access to the specification of a standard (a protocol, an interface, or a data format) used by other programs. Only then a working interoperable program can be developed and distributed to users. In passing, we note that access to source codes or binaries of such other programs is not necessary, as long as the standard specification is available.[9] By looking at (the number or weight) of the conditions on the access to standard specifications, it is possible to distinguish many different types of standards. We would like to identify the following two extreme types.

(1) **Closed standards.** Protocols, interfaces, and data formats are designed in private and subject to exclusive control. The control encompasses who may use, develop, and distribute programs that comply with a closed standard. This means that access to specifications of closed standards is restricted to certain persons only.

(2) **Open standards.** The participation in the design of protocols, interfaces, and data formats is not restricted, and no-one is precluded from using, developing, and distributing programs that comply with an open standard. This means that access to specifications of open standards is not restricted to anyone.[10]

9 Sometimes, standard specifications are accompanied by a reference implementation, that is a piece of source code of a program that uses the standard. The availability of the reference implementation is an additional aid in understanding of the specification and the implementation of the standard in other programs.

10 We present a more elaborate definition of open standards in Section 2.3.

As is the case in the software scene, between these two extreme types of standards we observe a spectrum of possibilities. We call this spectrum the *standards scene*. Later on, we will explain that any standard which does not meet the definition of an open standard as given in this thesis (see Section 2.3) is to be considered a closed standard, at least from the point of view of the Free Software users. Thus, in the remainder of the thesis we would like to downsize the spectrum of standards to the two extremes only.

1.1.3 Four combinations of software and standards

In the two subsections above, we split the main scene of play into a software scene and a standards scene. Each subscene consists of a spectrum with two extreme types. Based on these four extremes we may distinguish four combinations that will guide our research throughout the thesis. They are given in Table 1.1.

	Closed standards (CS)	Open standards (OS)
Proprietary software (PS)	PS CS	PS OS
Free Software (FS)	FS CS	FS OS

Table 1.1: Four combinations of software and standards

Theoretically, all of the above four combinations are equally possible. The proprietary software approach does not exclude a possibility that a proprietary program uses an open standard (PS OS). In such a case, although the development and distribution of such a program is restricted, the development and distribution of Free Software programs that are able to interoperate with such a program is not restricted. As a result, users can freely use such Free Software. However, many developers of proprietary software design it to use closed standards only (PS CS). In such a case, the ability of other developers to develop interoperable software is restricted. There are also restrictions in distribution of software that uses closed standards.[11] The use of such programs is restricted as well. Nevertheless, Free Software developers sometimes attempt to make their programs interoperable with other programs that use closed standards (FS CS).[12] But the most preferred way of developing Free Software is to design it according to open standards (FS OS). The distribution and use of FS OS is also the least-restricted one.

11 For example, such a distribution can constitute a violation of a patent material to the closed standard. See: Subsection 3.2.1.

12 We elaborate on such a situation in Subsection 3.2.2.

1.2 Actors and their audience

There are many persons that attempt to bring us closer to the world of seamless cooperation between computer programs and human beings. They play various roles in all of the four combinations of Table 1.1 by developing or distributing software. For the purpose of this thesis, it is desirable to take a closer look at these actors and to describe their roles. We will also pay some attention to their audience. We first discuss (1) actors and their audience in the software scene, and then (2) actors and their audience in the standards scene.

1.2.1 Actors and their audience in the software scene

Generally speaking, proprietary software (PS OS and PS CS) results in a strong dichotomy between the actors and the audience. Conversely, Free Software (FS OS and FS CS) blurs the line between the actors and the audience. Important actors in the Free Software scene are software communities. eGovernments play important roles both in the proprietary software scene as well as in the Free Software scene. The roles of eGovernments are important, mostly because of the impact that activities such as software procurement have on all other actors and the audience in the software scene.

Below, we discuss (1) actors and their audience of proprietary software, and (2) actors and their audience of Free Software.

(1) Actors and audience of proprietary software

The working of proprietary software is under exclusive control of a limited number of persons (copyright holders). Only they have access to source codes of this software and decide who may play an active role in its development and distribution, as well as who may use it. Conversely, users do not have individual control over proprietary software.[13] Their role is restricted to the consumption of binaries as delivered by the distributors. Certainly, this is not to say that a user demand for bug fixes and new features is not met. It rather means that users have to rely on copyright holders of a given piece of proprietary software to have their demand for properly working programs satisfied. So, we may definitely refer to all users of proprietary software as "audience". Consequently, only the copyright holders of proprietary software, and the developers and distributors appointed by them can be referred to as "actors".

We illustrate the above findings in Figure 1.1.

13 It is possible to provide examples of proprietary software that is delivered in a form highly customizable for users. Nevertheless, such customization is either performed by persons authorized by the copyright holder, or the copyright holder (not the user) controls the degree of the possible customization.

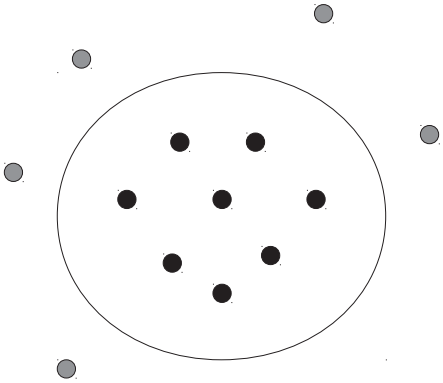

Figure 1.1: Actors and audience in proprietary software

In Figure 1.1 we see black dots illustrating actors that play an active role in the scene of proprietary software. They are the developers and distributors employed or otherwise contracted by the copyright holder (usually, the copyright holder is a firm). Grey dots illustrate users of the program, the consuming audience. The boundary between the former group and the latter group is solid. The users may not easily cross it, since they neither have access to source codes, nor they are allowed to develop or distribute the software by its copyright holders.

Some governments choose to introduce eGovernment based on proprietary software. However, government procurement of a proprietary program does not remove the control of the program from the hands of its copyright holder. In particular, the copyright holder may still control the development of this software and control who may become a distributor of this program. Consequently, the government on their own may not control the working of this program. Also, the government may not freely choose between various distributors of a given proprietary program. The government may only choose between the distributors appointed by the copyright holder. Thus, the government can only play a passive role as all other users of proprietary software.

(2) Actors and audience of Free Software

The line between actors and audience is much more blurred on the Free Software scene.[14] Copyright holders release this software from their exclusive control (although they retain copyright protection). They allow any user to step into an active role of a developer by eliminating bugs or including new features. Also, anyone is allowed to become a distributor of a given piece of Free Software. This means that both developers and distributors of this soft-

14 As of now, thousands of programs are being developed as Free Software. In August 2008 there have been approx. 85,000 projects under one of the most popular Free Software license, the GNU GPL v2, at a popular repository SourceForge (http://Sourceforge.net).

ware are generally self-appointed. So, everyone can become an "actor" in the Free Software scene. However, nobody is required to play such an active role. Thus, a user may remain passive and form a part of the "audience". Still, even members of the audience can entrust the roles of developers or distributors to any person of their choice.

Probably the most active actors of the Free Software scene are "hackers".[15] Generally speaking, hackers are technologically-savvy users who value the possibility of developing and distributing useful software tools without restrictions. They do so because they like to control the working of software and they are motivated to improve it, and to enable other users to use the improvements. Hackers are motivated for many different reasons. Some of the reasons are: (1) ethical reasons, (2) economic reasons, and (3) personal reasons. First, some hackers develop or distribute Free Software because they simply believe it is the right thing to do. Second, some hackers do so because they are getting paid for it.[16] Third, some hackers develop or distribute Free Software because it solves their problem (performs a particular task), because by doing so they gain peer recognition, or because they otherwise improve their social status in such a way.

Firms are also important actors in the Free Software scene. Firms do not exercise exclusive control over Free Software as it is the case with firms in the proprietary software scene, at least not directly. Still, firms contribute to the development of Free Software, as well as they distribute it. They are attracted to developing and distributing Free Software in particular by the fact that in such a way they support their business models. The term "Open Source Software" has been promoted to convey the technical superiority and commercial benefits of Free Software (see Section 2.2).

Many actors in the Free Software scene (including hackers and firms) work alone,[17] but some of them participate in communities. Communities are groups of actors who cooperate in fixing bugs and in including new features of a given program (see Section 2.4). Communities may also organize distribution of Free Software and provide guidance on its use. An important community has been originated by Stallman, under the name of "GNU Project".[18] Since the 1980s, many other programs have been developed and distributed as Free Software, not necessarily related to GNU. Most prominent examples of Free Software include, *e.g.*, the Linux Kernel, Debian

15 A hacker is properly defined as "a person who delights in having an intimate understanding of the internal workings of a system, computers and computer networks in particular." (*RFC1392*, at: http://rfc.net/rfc1392.html).

16 For example, according to some studies more than 70% of Linux kernel developers are getting paid for their work on Linux (See: The Linux Foundation, *Linux Foundation Publishes Study on Linux Development*, http://linux-foundation.org/weblogs/press/2008/03/31/linux-foundation-publishes-study-on-linux-development-statistics-who-writes-linux-and-who-supports-it/).

17 In Chapter 4 we refer to this as the "individual development and distribution of Free Software" or "individual exercise of the freedoms".

18 See: http://gnu.org, http://savannah.gnu.org.

GNU/Linux, Apache, or Mozilla.[19] Many Free Software programs have attracted large and well organized communities.

Outside of the communities there are other actors who at their choice may: (1) develop Free Software on their own, privately; (2) form a group of active developers outside of the existing community of a given program; (3) become an individual distributor of Free Software.

Apart from the above-described actors, there is also the audience in the Free Software scene. The audience is formed out of such users of Free Software who are not copyright holders, developers, or distributors of this software. Such users do not participate in software communities as well. The audience uses Free Software in a manner similar to proprietary software (by simply running the binaries on their computers, without paying attention to source codes). Still, there is a difference between the users who form the audience of Free Software and the users who form the audience of proprietary software. The difference is that the former have a choice whether to remain passive or become actors, while the latter do not have such a choice.

From the above we may conclude that users are not bound to play passive roles in the Free Software scene. Many users of Free Software actively develop the software or distribute it to other users. Such users play active roles, either individually (such as by becoming a hacker), by using a firm, or within a software community. We illustrate this in Figure 1.2.

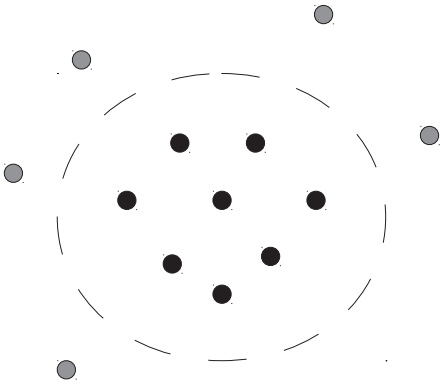

Figure 1.2: Actors and audience in Free Software

In Figure 1.2 we see black dots that illustrate users (*i.e.*, developers or distributors) who play an active role in the development or distribution of a Free Software program. They are, for example, individual hackers or firms. Some of them form a software community. Others develop or distribute the same program individually, outside of the community, or in a "competing" community. Grey dots illustrate passive users of the program (the audience).

19 See respectively: http://kernel.org, http://debian.org, http://apache.org, http://mozilla.org.

The boundary between the actors and the audience is dashed. Users may easily cross it, since source codes of the program are available to them for development or distribution.

The government may procure Free Software for use in eGovernment. As a result of such a procurement, control over Free Software is not given to any particular entity on an exclusive basis. The government and other users are allowed to continue using the same Free Software developed or distributed by actors of their free choice. The government could as well remain a passive user of Free Software, but they may also undertake software development or become distributors of this software to other users. In particular, the government can participate in software communities or develop Free Software on their own. Also, the government might stimulate proactive individuals to become developers or distributors. For example, the government can do so by collaborating with the communities in many different ways.

1.2.2 Actors and their audience in the standards scene

We start by recalling the two conditions necessary for the ability to control the working of a program. They are: (1) access to source codes, and (2) access to interoperability information, most preferably as included in specifications of standards. In Subsection 1.2.1 we analysed the actors and their audience in the software scene, *i.e.*, in the use of source codes. In this subsection we take a closer look at the actors and audience in the standards scene, *i.e.*, in the use of interoperability information (standard specifications).

Governments play an important role in the standards scene. In particular, government-procured software allows to use eGovernment services. In order to be able to use these services, users have to obtain programs that interoperate with programs used by the government.[20] Thus, whether the programs used in eGovernments are designed to interoperate using an open or a closed standard is a material factor.[21]

Below we focus on the roles of the actors that use Free Software (both FS CS and FS OS). We analyse two situations in the standards scene: (1) a situation wherein a Free Software program is required to interoperate with another program designed according to a closed standard (FS CS), and (2) a situation wherein a Free Software program is required to interoperate with another program designed according to an open standard (FS OS).[22]

20 A user of eGovernment services may be an individual, a firm, a public agency, a non-governmental organization, or another entity.

21 *Cf*. Table 1.1 and accompanying text.

22 In both of these situations, the other programs may be proprietary software or Free Software. The other programs may be also programs used by governments to provide eGovernment services.

(1) Actors and audience of closed standards

Closed standards are protocols, interfaces, and data formats that are exclusively controlled. Usually, the exclusive control is exercised by the firm that designed a standard (the "designer"). The designer may refuse to reveal interoperability information to the developers of a Free Software program. In such a case, Free Software developers could still be able to access this information using various techniques (we discuss them in detail in Subsection 3.2.2). If they succeed, it leads to the development of a Free Software program that uses a closed standard. Then, the Free Software developers and distributors may be referred to as actors. However, perfect interoperability is rarely reached in such a situation. Also, the development, distribution, or even use of programs that use closed standards can be restricted, because designers could impose conditions on the use of these standards (*e.g.*, by enforcing a patent). So, in practice only the designers of closed standards are actors. The designers may allow other persons to become actors, but they may also decide to put everyone else in the position of the audience.

For eGovernment, some governments procure software designed to use a closed standard. The government does not become an actor in such a way, since the control over the working of programs which use that standard remains in the hands of the designer of the standard. In particular, the designer of such a standard retains exclusive control over who is authorized to develop programs which are able to interoperate using that standard. Obviously, users have to procure programs that interoperate with eGovernment software, since otherwise they may not use the services of eGovernments. But only the developers authorized by the designer of closed standards used in eGovernment software can develop properly interoperable programs. Sometimes, only the authorized distributors can distribute such programs. This allows the designer to stimulate users to procure selected software only.

In some cases eGovernments based on closed standards may even (1) effectively isolate a particular distributor from its competitors, (2) direct users to play the role of passive consumers, and (3) prevent the developers of Free Software from developing this software. In other words the use of a closed standard in eGovernments contributes to making the audience consisting out of everyone but the designer of this standard or persons appointed by the designer.

(2) Actors and audience of open standards

Open standards are not exclusively controlled. By definition, no-one is able to restrict anyone in developing their programs to use an open standard. In particular, Free Software developers that intend to make their programs interoperate using open standards may do so freely. Also, Free Software distributors may then distribute such interoperable programs without restrictions that may apply to closed standards. When using open standards a high degree of interoperability is possible. This means that anyone may become an actor in the scene of open standards. No-one, however, is required to play

an active role and everyone may remain the audience by simply using software designed according to open standards (either proprietary or Free Software).

If the government chooses software based on open standards for eGovernment, both proprietary and Free Software developers may continue to develop programs that interoperate with such software. Anyone could become a distributor of such interoperable Free Software either for the government or for users of eGovernment services. Certainly, the development of proprietary software that interoperates with eGovernment remains possible, and the distributors of proprietary software are still able to distribute it to their users. Consequently, users may use eGovernment services by choosing between Free Software and proprietary software. Additionally, anyone (including the government) may become an actor by undertaking development or distribution of programs that interoperate with eGovernment and that are Free Software.

1.3 Problem statement

Below we formulate our Problem Statement. We start by recalling that we envisage a world where humans and computer programs cooperate. We aim at a seamless cooperation which in particular implies that computer programs should work efficiently and properly. In order to reach this goal, we believe that the working of programs should be controlled. For this purpose we identified two necessary conditions: (1) access to the program's source codes and (2) access to the specifications of the standards used by the program to interoperate with other programs. A person who meets these two conditions is able to bring us closer to the world envisaged. This leads to an important question, namely: "*Who* is allowed to control the working of computer programs?"

(1) Two extreme approaches

In Table 1.1 we presented four combinations of software and standards: (1) FS OS, (2) FS CS, (3) PS OS, and (4) PS CS. In each combination, the control over the working of computer programs is assigned differently. Under the FS OS, attempts are made to give the control to every user. Under the remaining three combinations control is retained by certain individuals. The latter three combinations differ as to the degree of control that such individuals have. Yet, what they have in common is that under each of them the control cannot be exercised by every user. So, for the formulation of our Problem Statement, we will consider the three latter combinations together, which brings us to two extremes: (1) the FS OS (henceforth called "the Free Software approach"), and (2) the three other combinations together (henceforth jointly addressed as the "proprietary approach").

(2) The Debate

There has been a fierce debate about which of the two approaches is better. Various participants in the debate have presented their positions. Stallman's position is an example of the set of positions that oppose the proprietary approach.[23] It may be summarized in its considering the proprietary approach as morally unacceptable. A second example of positions that oppose the proprietary approach is the position of the advocates of "Open Source Software" (as we explain below, Open Source Software can be considered as closely related to Free Software). In a nutshell, they consider that the proprietary approach should not be followed because it is less efficient.[24] Supporters of the proprietary approach present their positions as well. A popular position amongst them is that most users only want to use software which serves their purposes of the moment, and they have no need to control their working. According to this position, the proprietary approach in itself can bring us closer to the world where humans and computer programs cooperate seamlessly.

(3) Our Position in the Debate

We describe our position below. It takes into account the following three issues: (1) Stallman's rejection of the proprietary approach on moral grounds, (2) negative evaluation of the efficiency of the proprietary approach made by the Open Source advocates, and (3) assessment of the needs of users as made by the supporters of the proprietary approach.

Our position is as follows. Ad (1) The proprietary approach may be morally unacceptable to many people. Such morality may lead them to abandon the proprietary approach. If so, then that possibility should be adequately covered by the law. In our opinion, everyone should be allowed to follow their own moral standard as long as they do not act against the law. Ad (2) Free Software programs are often of good quality. This means that the Free Software approach is generally efficient. We believe that there is no evidence for the proprietary approach to be unable to deliver good quality software for a comparable price. Here we remark that the economic competition, if any, should also be covered by the law. Ad (3) Users who wish to control the working of software do not constitute a majority. But the law should cover adequately the goals of all users, *i.e.*, the goals of laymen who only want to use software "as is", as well as the goals of the knowledgeable persons (specialists).[25]

---

23 See, *e.g.*, R. M. Stallman, *The GNU Project*, at: http://www.gnu.org/gnu/thegnuproject.html.

24 See, *e.g.*, Open Source Initiative, *Open Source Case for Business*, at: http://www.opensource.org/advocacy/case_for_business.php.

25 Notably, even users who do not wish to control the working of software often prefer to use Free Software above using proprietary software. For example, while proprietary software still holds the majority of the market for desktop operating systems and office applications, Free Software is often chosen in the server market and as the basis for web applications. Free Software is also quite popular amongst manufacturers of embedded devices.

In summary, we state the following. Many Free Software programs bring us closer to the world as envisaged. The proprietary approach is aiming to do the same. Users may have different points of view depending on their technical qualifications, moral attitude, *etc*. Our position is that everyone should be allowed to choose an approach which they consider best for themselves. For this purpose it is necessary that both approaches are sufficiently protected by an adequate regulatory framework.

(4) Point of Departure: The Free Software Approach

In the thesis, we will analyse the protection of the Free Software approach. We find such an analysis appealing for the following two reasons.

First, we are not aware of any material threats to the proprietary approach. Indeed, there exist various laws that enable individuals who wish to control software exclusively to do so. The legal tools available to them definitely succeed in preventing the users of proprietary software to exercise the degree of control which is proposed under the Free Software approach. So, there is little risk that the proprietary approach will be displaced by the Free Software approach in the foreseeable future. Consequently, there is no apparent need for a legal research directed at the preservation of the proprietary approach.

Second, the Free Software approach faces many more threats than the proprietary approach. The preservation of the Free Software approach requires material efforts. Examples of threats mentioned above are software-related patents or eGovernments based on closed standards. Although there is already an impressive body of literature on the Free Software approach, such problems have not been addressed to an extent necessary to guarantee that a choice between the two approaches will exist in the long run.

(5) The Research Goal

Our concern is that the regulatory framework for protecting the Free Software approach needs further scrutiny, especially under the current conditions, where governments (*e.g.*, Germany, the Netherlands) specify policies supporting it and yet often find them difficult to implement. This issue leads to the following question: "To what extent is it possible to give every user control over the working of computer programs?" An answer to such a question could provide valuable information about limitations and restrictions of the current regulatory framework of Free Software.

The information could then be used, for example, in the development of an improved framework, which would make it easier to follow the Free Software approach. In particular, this would facilitate the governments to substantiate their policies. It would also benefit everyone in the pursuit of a world of seamless cooperation between computer programs and humans. We believe that an improved framework would protect the Free Software approach from threats which often make the proprietary approach the only realistically possible approach. So, we find the above question accurate for the purpose of the formulation of our Problem Statement. Moreover, we

stress that the development of an improved framework is to be considered as the Research Goal of this thesis.

(6) The Free Software Definition
For the formulation of a Problem Statement that can be dealt with in a scientific way towards the Research Goal, we need to focus on a clearly defined notion of Free Software. In the preceding sections, we already provided a general understanding of this notion, but not a precise definition. There are two precise definitions that deserve consideration: (1) the "Free Software Definition" (hereinafter "FSD"), and (2) the "Open Source Definition" (hereinafter "OSD"). In this thesis we decided to follow the FSD. Below, we perform a brief assessment of both definitions, and explain why FSD is better suited for the purpose of our research. A detailed analysis of the FSD and the OSD is presented in Chapter 2.

The FSD has been drafted by Stallman.[26] It specifies the following four freedoms of software users (we only quote the parts most relevant for our study).

Freedom 0: *"The freedom to run the program, for any purpose."*

Freedom 1: *"The freedom to study how the program works, and adapt it to your needs. (...)"*

Freedom 2: *"The freedom to redistribute copies so you can help your neighbor."*

Freedom 3: *"The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. (...)"*[27]

If a given program is subject to all the freedoms specified above, it is Free Software. Otherwise, the program is proprietary software. We find that the FSD is an accurate definition for the following two reasons.

First, user freedoms as specified in the FSD mean in particular that users are able to modify a Free Software program by removing bugs, including more features, and designing or modifying it in such a way that it interoperates with other programs. The four freedoms allow users also to develop programs, and to distribute them, so that other users are able to use them as well. It is hard to imagine what else could be necessary for users to control the execution of software.

26 R.M. Stallman, Free Software Definition, *op. cit.*
27 *Id.*

Second, in Chapter 2 we conclude in particular that the FSD requires that users are allowed to exercise all activities covered by copyright. Indeed, if a user were not able to exercise all these activities, the control would still require the copyright holder's consent. So, any degree of control lesser than the degree specified in the FSD would mean in fact that the control is not given to users.

(7) The Open Source Definition

The OSD formulates certain requirements for a program to be considered Open Source. But Open Source programs and Free Software programs do not constitute separable classes. Both definitions contain essentially the same or similar requirements, although they use different wording. They both have been applied to scrutinize model software licenses with results that are to a large extent similar (the most popular Free Software licenses are also considered Open Source licenses).[28] As a result, most (if not all) programs that are considered Free Software are at the same time considered Open Source.[29] So, it seems that we could as well adopt the OSD as the definition that guides our research. But the FSD is better suited for the purposes of legal research. It specifies freedoms. "Freedom" is a legal notion, capable of being analysed using legal methodology, which is the main methodology used in this thesis.

(8) Consequences of following the Free Software Definition

The reader should be aware that by following the *Free Software Definition* we do not necessarily adopt Stallman's moral position as mentioned above. Neither do we explicitly reject or identify ourselves with the position of Open Source advocates. Here we reiterate that both the Free Software approach and the proprietary approach are able to bring us closer to a world where humans and computer programs cooperate. Our position is that everyone should be allowed to choose an approach which they consider best for themselves. Precisely speaking, the fact that we follow the Free Software Definition means that we consider as "Free Software" only computer programs over which users can exercise control as specified by Stallman.

---

28 For details the reader should refer to Chapter 2.

29 It might be the case that some software licenses that do not meet the FSD can still satisfy the OSD (allegedly, the OSD requires that a lesser degree of control is given to users). According to Stallman, the OSD is "a little looser in some respects, and they have accepted a few licenses that we consider unacceptably restrictive of the users." (R. M. Stallman, *Why "Free Software" is better than "Open Source"*, at: http://www.fsf.org/licensing/essays/free-software-for-freedom.html). So, there is some risk that if we followed the OSD in our research we would find a given level of protection of the Free Software approach adequate, while actually users would not be able to exercise control over software at that level. Although we do not find arguments that this is a material risk (see Chapter 2), it is an additional reason to follow the FSD in this thesis.

(9) Two Conditions of User Freedoms

The reader is obviously aware that, as a default legal rule the control over who may use, develop, and distribute software lies in the hands of the copyright holders. *Ceteris paribus*, the existence of the freedoms is possible, only if the copyright holders decide to change the default arrangement. Therefore, their affirmative action (consent) is required in order to allow users to exercise the freedoms, in particular to develop or distribute a given piece of software.

Here, a new question arises, namely whether the consent of copyright holders to software is sufficient for users to exercise the freedoms. Stallman notes correctly that access to source codes (a source code is a copyrightable expression of software) is a condition for Freedom 1 and Freedom 3.[30] However, given our previous findings, a program may be developed to interoperate properly with other programs and then it may be distributed without restrictions if both (1) its source codes and (2) the specifications of standards used by other programs with which the program has to interoperate are accessible.[31] It means that the relaxation of private control over software (in particular over the program's source codes) is only the first necessary condition of user freedoms. The relaxation of private control over standards (in particular the interoperability information expressed in standard specifications) is the second necessary condition of user freedoms. So, we will analyse the protection of user freedoms by analysing whether both these conditions are satisfied and to what extent.

(10) Protection of User Freedoms

User freedoms are brought into existence by the copyright holders and designers of standards who follow the Free Software approach by relaxing their control over software and standards, respectively. The freedoms continue to exist as long as they are properly protected over time. The protection necessary to sustain a situation wherein every user is able to control the working of Free Software. In other words, the protection is necessary for the Free Software approach to be possible in the long run. Whether the freedoms are sufficiently protected depends on the rules that regulate the use, development, and distribution of software. These rules (and the relations between them) can be arranged into a regulatory framework. In this thesis, we call such a framework "the regulatory framework of Free Software", or "the framework" in short. We reconstruct a model of the current regulatory framework of Free Software in Chapter 3.

(11) The Focus on Software Communities and eGovernments

The model of the current regulatory framework can be used in a search for an answer to the question: "To what extent is it possible to give every user

30 R.M. Stallman, *Free Software Definition*, *op. cit*.

31 By "accessible" we mean here not only available without limitations but also free from any legal restrictions.

control over the working of computer programs?" But the extent of research necessary to perform such a task outgrows the resources gathered for the preparation of this thesis. At the same time, such a research would not be focussed on issues that are materially relevant. To make our task practicable, we need to formulate a Problem Statement tailored to circumstances, in which the Free Software approach experiences restrictions and limitations of material importance. From this point of view, software communities and eGovernments are of particular interest. As follows from previous sections they affect the use, development, and distribution of Free Software to a large extent. So, in the formulation of our Problem Statement we focus on the relations between software communities, eGovernments, and user freedoms.

Software communities evolve as a result of individual decisions of various actors. By exercising their exclusive rights, using contractual freedom, and the freedom of association, the holders enable other individuals to form communities. Within the communities, the participants are guided and supported in their efforts to use, develop, and distribute Free Software. This is done by organizing relations between copyright holders and other actors. By organizing these relations, the communities affect the protection of user freedoms. Essentially, software communities influence the Free Software framework. The role of the communities will later be analysed in detail in Chapter 4.

eGovernments affect the relations between all actors in the scenes of software and standards and their audience. Holders of rights to software, holders of rights to standards, software developers, software distributors, and users are all affected by eGovernments. They affect software communities as well. Government procurement of software has the most important impact, but other activities of eGovernments also influence the ability of all the actors to control the working of programs. Consequently, eGovernments affect the protection of user freedoms and they influence the Free Software framework. The role of eGovernments will be analysed in detail in Chapter 5.

(12) Formulation of the Problem Statement

So, in this thesis, we undertake an analysis of the roles of software communities and eGovernments in the protection of user freedoms. This leads us to a twofold Problem Statement (PS).

PS 1: *What are the relations between user freedoms and software communities?*

PS 2: *What are the relations between user freedoms and eGovernments?*

These problems are intriguing but difficult to answer on their own. Therefore, we formulate three additional Research Questions in the next section.

1.4 Research questions

Our Research Goal is to develop a new, improved regulatory framework that would be capable of resolving the inefficiencies in the protection of user freedoms identified by analysing the Problem Statement. To achieve this goal, we formulate the following three Research Questions (RQs) that guide us throughout the analysis.

RQ 1: *In what way do software communities affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?*

RQ 2: *In what way do eGovernments affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?*

RQ 3: *How to improve the regulatory framework so that it adequately protects user freedoms, as articulated by Stallman, in the world of software communities and eGovernments?*

We analyse the Problem Statement using a model of the current regulatory framework, within which *software communities* and *eGovernments* operate. We follow Stallman's Free Software Definition in the analysis. In Section 1.6 we indicate in which chapter the Research Questions are addressed.

1.5 Research methodology

In order to answer the Problem Statement it is necessary (1) to reconstruct the current regulatory framework of Free Software in such a way that it can deal with our Research Questions. Then, it is necessary (2) to analyse how software communities and eGovernments affect the ability of the framework to protect the freedoms. Thereafter, we have (3) to identify the elements of the framework that protect the freedoms inadequately, and (4) to design the necessary amendments and propose specific modifications to the framework.

The framework consists of (1) rules that regulate the use, development, and distribution of Free Software, and (2) the relations between the rules. So, we need to identify the rules and the relations by analysing their sources. In this thesis, we adopt a broad concept of regulation that may be the source of rules and relations between them. We identify the following four means of regulation, as specified by Lawrence Lessig:[32] (1) the law, (2) the architecture, (3) the norms, and (4) the market. First, the law regulates by executing sanc-

32 Lawrence Lessig, *The New Chicago School*, 27 The Journal of Legal Studies 661 (June 1998).

tions for infringing rights or not performing obligations.[33] Second, the architecture consists of natural limitations, as well as of restrictions created by humans. The architecture regulates by making some actions impossible or impracticable.[34] Third, the norms regulate by using non-legal sanctions, as they are the creation of a social culture.[35] Fourth, the market regulates by the price mechanism.[36] We use these four concepts in our research methodology. Moreover, we adopt from Lessig the observation that regulation takes place both directly through these four means and indirectly by interactions between the means themselves.[37]

In the thesis we concentrate mostly on the rules created by the first regulator, the law. Thus, the main methodology used to answer the research questions is the examination of sources of law. The objects of this examination are primary and secondary sources of law that regulate access to source codes and to standards to the extent this access is necessary to use, develop, and distribute software. The most relevant body of law is copyright law. However, we also refer to many other laws, because they affect the protection of the freedoms in some other way. In the analysis of a particular body of law, we concentrate on the black letter law (statutes and case law), and we also refer to legal commentaries and other examples of jurisprudence. We restrict our research to the laws of the European Union and Poland, but we often refer to the American law for a clear comparison. The research methodology uses logical legal reasoning to infer legal rules that form the regulatory framework and relations between the rules.

Our research is extended beyond the legal research in order to determine relations between legal rules and the rules that are created by the other three regulators: the architecture, the norms, and the market. Thus, we analyse the regulating power of law in relation to the regulating powers of: (1) the architecture of software and standards; (2) the norms that have evolved in the development of software and standards (in particular in the hacker culture); and (3) the markets of software and standards (in particular under the theories of transaction costs, network externalities, lock-ins *etc.*). In the thesis, the research in these three other means of regulation is directed mostly at the identification of the practical impact that legal rules of the framework may have on the freedoms under various circumstances.

In summary, our research methodology will consist of: (1) reconstructing a model of the current regulatory framework by using logical legal reasoning in the analysis of primary and secondary sources of law, as well as by analysing non-legal regulators of software and standards; (2) using the model to identify inefficiencies of the current regulatory framework from

33 *Cf*. *id*. at 662.
34 *Id*. at 663.
35 *Id*. at 662.
36 *Id*. at 663.
37 *Id*. Figure 2 at 667, and accompanying text.

the point of view of user freedoms as exercised in the world of software communities and eGovernments, and (3) formulating improvements to the framework.

1.6 Structure of the thesis

In order to answer the Problem Statement, we start by focussing on the Research Questions. The following structure guides our research. In Chapter 2, we provide definitions and basic notions that are used throughout the analysis. In Chapter 3, we reconstruct a model of the current regulatory framework by identifying the main sets of rules and relations between them. In Chapter 4, we focus on software communities and we formulate an answer to the Research Question 1. In Chapter 5, we focus on the eGovernments and we formulate an answer to the Research Question 2. In Chapter 6, we use the results of the analysis to formulate a proposal for an improved framework, which constitutes an answer to the Research Question 3. In Chapter 7 we provide the conclusions of the thesis and speculate on the provisional impact of our proposed framework. Moreover, we formulate directions for further research.

2 Definitions and basic notions

This chapter contains definitions of the most important terms and presentations of the basic notions used throughout the thesis. All of them were already briefly introduced in the previous chapter. We start elaborating on "Free Software" (in Section 2.1) and on "Open Source Software" (in Section 2.2); it is followed by an explanation of the relationship between these two notions. Thereafter, in Section 2.3 we elaborate on "open standards" and in Section 2.4 on "software communities". Finally, in Section 2.5 we discuss "eGovernments".

2.1 Free Software

Free Software is software that is available to any interested party together with the rights to control the working of the software. In order to allow for an effective control, the rights to Free Software encompass source codes, not only the binaries.[38]

Three practical implications of granting all these rights and making source codes available are as follows.

(1) Users may use a Free Software program.
(2) Users may actively develop it by removing its bugs and including new features.[39] They may undertake the development either in a software community, or outside it.
(3) Users may become distributors of Free Software by sharing it with other users. They can share non-commercially, but they can also offer on the market both the program and any additional services that other users might want to bargain for.

38 As we already explained, a source code is a human-readable expression of a program that allows for its effective modification. It also allows to study the program and understand its structure. For the purpose of using a program it has to be translated into a machine-readable object code (also referred to as a "binary"). Binaries are impracticable to study and modify; this makes access to source codes necessary for the purpose of modification. The latter constitutes a major part of a software development process. The need for modifications is caused by (1) the existence of errors ("bugs"), (2) the lack of necessary features, as well as (3) the need to provide for interoperability with other programs.

39 The may also provide means for interoperability of the program with other programs, assuming that apart from source codes they also have the necessary interoperability information (standard specifications).

There are the following three common misinterpretations.

(1) Free Software should not be mistaken for "non-commercial". The defining criterion of Free Software is the scope of the user's rights, not price.
(2) Free Software should also not be mistaken for "freeware", which is proprietary software given away gratis. Free Software may cost a price and it comes with certain significant rights that users of freeware do not obtain.
(3) Free Software should not be mistaken for "public domain" software, which is software not covered by copyright. Free Software is usually copyrighted, although licensed under specific terms. Precisely speaking, software released into the public domain together with its source codes is an example of Free Software, but not all Free Software is public domain software.

Stallman[40] has defined *a minimum scope* of rights that have to be granted to users of a program in order for it to be considered Free Software. This minimum scope embraces the four freedoms that have been expressed in the formal Free Software Definition ("FSD"),[41] and that we already listed in Section 1.3. We repeat them here for a clear reference.

Freedom 0: *"The freedom to run the program, for any purpose."*

Freedom 1: *"The freedom to study how the program works, and adapt it to your needs. (...)"*

Freedom 2: *"The freedom to redistribute copies so you can help your neighbor."*

Freedom 3: *"The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. (...)"*[42]

In order to understand the FSD precisely we have to know what the subject matter is, to which its requirements apply (see Subsection 2.1.1). Then, we have to know what these requirements exactly are (see Subsection 2.1.2). Afterwards, we have to identify the addressees of the requirements (see Subsection 2.1.3). Finally, we formulate conclusions on the FSD (in Subsection 2.1.4).

40 R.M. Stallman, Free Software Definition, *op.cit.*
41 *Id.*
42 *Id.*

2.1.1 Subject matter of the Free Software Definition

The subject matter of the FSD is software. Precisely speaking, the FSD extends to both source codes and binaries. Namely, it makes access to source codes a necessary condition of the freedoms (Freedom 1 and Freedom 3). It also requires that users should be free to redistribute binaries as well. Conversely, the FSD is silent on the specifications of protocols, interfaces, and data formats. Thus, a program may be Free Software even if its users do not have access to specifications of standards used by this program to interoperate with other programs.[43]

Given the fact that both (1) access to source codes and (2) access to specification of standards are necessary for the protection of user freedoms, the satisfaction of the requirements of the FSD by a program is only the first (albeit necessary) step on the way towards the protection of the freedoms of users of that program.

2.1.2 Requirements of the Free Software Definition

The requirements of the FSD can be divided into (1) positive requirements, (2) negative requirements, and (3) optional requirements.[44] Positive requirements are *obligations* to include certain items for a program to be Free Software. Negative requirements are *prohibitions* to introduce certain items in relation to a Free Software program. Optional requirements are *permissions* for certain items as long as they are formulated in specific way, but nothing is required if they are absent. Below, we analyse all three types of requirements of the FSD.

(1) Positive requirements

We identify the following positive requirements. The FSD states that the freedoms should apply to every user. It also requires that the freedoms are irrevocable "as long as you do nothing wrong". Additionally, the FSD specifies activities that should be free to undertake for users of a program, for the program to be considered Free Software. These activities are: (1) running, (2) studying, (3) adapting, (4) redistributing, (5) improving, and (6) releasing.[45]

---

43 A separate issue is whether it is practically possible to develop a Free Software program without access to such specifications. Usually, it is either impossible, or the resulting program cannot interoperate properly with other programs.

44 The distinction between positive, negative, and optional requirements is for the convenience of the reader only. We are aware of the fact that any requirement which is here presented as positive could be expressed in a way that makes it a negative one, and vice versa.

45 In the whole FSD document close synonyms of these words are also used. For example, it also mentions "using", "modifying", "publishing", "changing", "copying", "selling". In this thesis, for the sake of clarity, we will consider only the six activities mentioned in the list of the freedoms as covered with the FSD. These are: (1) running, (2) studying, (3) adapting, (4) redistributing, (5) improving, and (6) releasing.

Not surprisingly, according to the FSD all these six activities should be free, not just some of them.

A question arises how these requirements translate into legal notions. The requirement that the freedoms should apply to every user means that every user should have rights specified in the FSD. The requirement that the freedoms be irrevocable "as long as you do nothing wrong" means that users should have their rights terminated only upon a breach. A breach is possible only if there are any obligations that can be breached. We note that the FSD does not specify exactly what these obligations are and what constitutes their breach.

The translation of the requirements that the six activities should be free for users requires a more elaborate analysis. According to copyright law, it is possible to restrict (at least) the following three activities: (1) copying, (2) distribution, and (3) modification of software.[46] So we have to determine what the relations are between the six activities that the FSD requires to be free to undertake by users and the three activities that may be restricted using copyright. We observe the following relations.

(1) *Running* software typically involves at least *copying*.
(2) *Studying* software is theoretically possible without undertaking any activities covered by copyright law. However, efficient *studying* of software usually involves *running* it, which brings us back to point (1) above and involves *copying*. Additionally, if source codes of a program are not available, the program may have to be decompiled to allow a person to study it efficiently. Decompilation involves *copying* and *modification*.
(3) *Adapting* and *improving* software requires *modification*, and sometimes also *copying*.

46 Generally, there are two approaches in copyright law worldwide. Under the first approach all imaginable activities with regard to the copyrighted work are subject to exclusive right. Under the second approach, the exclusive rights cover only activities enumerated in the respective law. The Software Directive includes in the exclusive copyrights of programs: reproducing, altering, distributing (to the public) including renting. The word "include" in its art. 4 together with reference to Berne Convention in Art. 1 allows to conclude that additional activities can be covered by exclusive rights by the Member States, but otherwise both the Software Directive and Berne Convention follow the second approach. Polish Copyright Act Art. 74.3 that implements the Software Directive, does not contain an open list, but according to some authors it should be interpreted to include activities not expressly mentioned therein (See *e.g.*, J. Barta, R. Markiewicz, Prawo autorskie [Copyright] (Wolters Kluwer 2008), 133). At the same time, the Polish Copyright Act expressly extends the protection granted in the Software Directive by including all kinds of lending in exclusive distribution rights (whereas the recitals of the Software Directive expressly leave public lending outside of its scope); this can be regarded as an implementation of the Rental Directive. Under U.S. Copyright Act software-related copyrights cover: reproducing, preparing of derivative works, distributing. For sake of simplicity, in this thesis we assume that copyright covers copying, modifying, and distributing, which generally corresponds to the scope of copyright under all laws known to us.

Towards_an_Improved_def.indd 46 23-02-2010 10:49:15

(4) *Redistributing* and *releasing* software constitutes a form of *distribution*. Sometimes it may be possible to release a program without distributing it, but we find it hardly possible without *copying*.[47]

We illustrate the relations between the six activities covered by the FSD and the three activities covered by copyright in Figure 2.1.

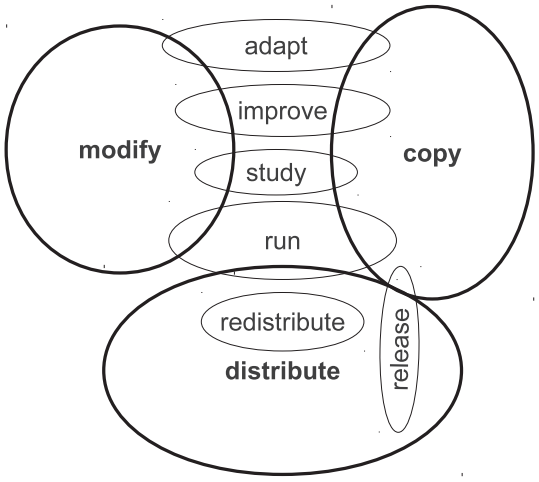

Figure 2.1: Relations between the FSD and copyright law

In Figure 2.1 thick circles represent aggregates of activities that involve copying, modification, or distribution (all three aggregates covered by copyright). Thin circles represent activities that involve running, studying, adapting, redistributing, improving, and releasing (all six aggregates covered by the FSD). Aggregates covered by copyright (copy, modify, distribute) do not have any overlap between themselves, but the six activities covered by the FSD do overlap with the three aggregates. Notably, under some circumstances (which we generally find unlikely), a user could undertake the running, studying, adapting, or improving of software without involving copying, modifying, or distributing. Conversely, we are not aware of any type of redistribution or release that would not involve any activities covered by copyright.

It follows that each of the six activities covered in the FSD usually requires at least one of the activities that may be restricted using copyright. It also follows that the exercise of all six activities covered in the FSD (*i.e.*, the exercise of all user freedoms) requires the exercise of all three activities covered by copyright. This means that if we restrict at least one of the activities covered by copyright the users are not able to exercise all six activities covered by the FSD.

47 A program may be released as a web application, which is not distributed. However, in such a case, the program has to be copied on a server.

Here, we may conclude that the positive requirements of the FSD can be translated into legal notions in the following way. A program is Free Software if every user of the program has rights specified in the freedoms. These rights can terminate only upon a breach of some (unclear) obligations. The rights referred to in the FSD should allow users to exercise all six activities mentioned therein. Using the copyright notions of (1) copying, (2) modifying, and (3) distributing one can conclude that the rights required by the FSD for users are copyrights. In other words, the FSD requires that users are free to exercise all activities covered by copyright. Nevertheless, the FSD does not require to release software from copyright protection, such as by transferring it to the public domain. In particular, the FSD does not forbid exercising one's copyright in software by licensing. Indeed, the FSD has been applied in practice by the Free Software Foundation (hereinafter the "FSF")[48] to scrutinize whether various software licenses meet the criteria for Free Software. As a result, the FSF has held many different licenses as FSD-compliant. Programs released under these licenses are Free Software.

(2) Negative requirements

The FSD generally prohibits to impose restrictions on Free Software. Particular attention is given to contract restrictions. The basic rule of the law of contracts is the freedom of parties. It means that there are almost no limitations on the restrictions that a party to a contract may impose on the other party, provided that there is an agreement. Consequently, the FSD states that it is impossible to list all possible contract restrictions and it reserves to the FSF the right to scrutinize each case separately. According to the FSD, "[i]f a contract-based license restricts the user in an unusual way that copyright-based licenses cannot, and [such a way] isn't mentioned here as legitimate, we will have to think about it, and we will probably decide [that the license] is nonfree." So, we have to note that the list of negative requirements in the FSD is not exhaustive, it contains examples only.

We identify the following negative requirements explicitly indicated in the FSD. The FSD prohibits to impose three conditions on the exercise of the freedoms: (1) payment, (2) notification, and (3) export control. First, under the FSD the payment for Free Software cannot be used to extend the rights of the users who paid for Free Software as compared to the rights of users who received it without payment. Remarkably, this does not prohibit selling Free Software or otherwise develop or distribute it against remuneration. The FSD explicitly states that commercial use of Free Software has to be always allowed. Second, the FSD requires that users have to be allowed to exercise their freedoms without having to notify anyone. Third, the FSD does not

48 The Free Software Foundation is a non-profit organization established by Richard M. Stallman. It promotes Free Software and it maintains the FSD. In particular, it scrutinizes various software licenses in order to determine whether they are compliant with the FSD. See, http://www.fsf.org.

allow to demand that the user observes any export control regulations to be able to exercise the freedoms.[49]

(3) Optional requirements

We identify four optional requirements of the FSD. First, the FSD allows to demand that the users distribute a Free Software program in a specific way, as long as they "don't conflict with the central freedoms" (we would like to refer to this requirement as "distribution rules"). Second, the FSD allows to demand that users package modified versions of Free Software in a particular way. However, this cannot block the freedom to release modified versions (Freedom 3) (we would like to refer to this requirement as "packaging rules"). Third, the FSD allows to demand that if a program is made available in a certain way, it also has to be made available in some other way (we would like to refer to this requirement as "publishing rules"). Fourth, the FSD allows to demand from a user who distributes a modified version to send a copy to a previous developer or to provide identification (we would like to refer to this requirement as "reporting rules").[50]

The FSD permits to impose the above mentioned rules provided some limits are observed. However, the FSD does not specify where exactly these limits are. Most probably, the limits have to be interpreted from the overall goal of the FSD and from the practice of its application. First, the whole phrasing of the FSD suggests that its goal is to safeguard the interest of users. This means that any ambiguities should not be used to decrease user freedoms. Second, from the licenses approved by the FSF as FSD-compliant it follows that in practice it is not possible to impose any material obligations on users.[51] We may conclude from the above that optional requirements are of minor importance. User freedoms have to remain practically unaffected by them, otherwise a program could not be called "Free Software".

2.1.3 Addressees of the FSD

A natural question that arises here is: to whom are all the requirements of the FSD addressed? In other words, who is required to observe that some items are present (positive requirements), who is not allowed to restrict users (neg-

49 The FSD notes that this would not make the export control regulations inapplicable. It would only allow users, who are outside of the jurisdictions where these regulations apply, to exercise their freedoms.

50 A careful reader should ask how can such reporting rules be consistent with the negative requirement to be allowed to exercise the freedoms without having to notify anyone. The FSD does not provide a clear guidance on this issue.

51 Free Software licenses usually require to provide users with certain information (copyright notices, license text), and pass through liability limitations and warranty disclaimers. Some licenses contain also so-called *copyleft* clauses, which require not to restrict user freedoms and provide them with source codes. For a good overview of notice requirements in various Free Software licenses see: Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 83 *et. seq*.

ative requirements), and who has to observe certain limits while demanding some performance from users (optional requirements)?

Obviously, all these requirements are addressed to copyright holders. However, there are many other entities that are able to restrict users in a way contrary to all the requirements of the FSD. For example, some Free Software distributors conclude contracts whereby users undertake to use certain short-listed software only, or to refrain from modification or distribution.[52] The distributors can alternatively attempt to restrict users by making a Free Software program a part of a device or a service (*i.e.*, by using the architecture). Also, some patent holders could enforce their software-related patents against users of Free Software, restricting their freedoms as a result. However, the FSF has not declared that programs subject to such third-party restrictions are non-Free Software. The FSF makes the FSD applicable only to licenses that regulate a relationship between users and copyright holders. So, only copyright holders are the addressees of the requirements of the FSD. We illustrate this in Figure 2.2.

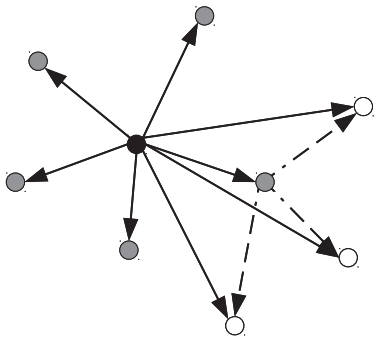

Figure 2.2: Addressees of the FSD

In Figure 2.2 we see a black dot that represents a copyright holder to a Free Software program (licensor). Other dots (grey and white) represent users bound by the license granted by the licensor and, possibly, with some contract restrictions that accompany the license. These relations with the licensor are represented by solid arrows and only these relations are regulated by the FSD. As we already explained in the previous chapter, some users (grey dots) become software distributors and distribute Free Software to other users. Users (white dots) that obtain a Free Software program from distribu-

52 See: Slashdot, *Hans Reiser Speaks Freely About Free Software Development*, at: http://developers.slashdot.org/article.pl?sid=03/06/18/1516239&tid=156&tid=11. ("You do all understand that while the GPL doesn't permit tying by license, distros have now moved to using threats of invalidating support contracts to achieve the market leverage they need to exclude competitors, yes? By doing this they can exclude mainstream official kernels from being used, exclude rival filesystems, exclude whatever might lead to less customer lockin.")

tors still remain in a direct relation with the licensor.[53] They are also in relation with the distributors, usually based on a contract concluded with the distributor, based on an additional distributor's license, or because they use the distributor's service or device. The relations with distributors are represented using dashed arrows. None of the user-distributor relations results in the program becoming non-Free Software. The FSD does not regulate the relations between distributors and users.[54]

Here, we may conclude that all requirements of the FSD are addressed to copyright holders only. They are not addressed in particular to distributors or other third parties such as patent holders.

2.1.4 Conclusion on Free Software Definition

From the above analysis we may conclude that the subject matter of the FSD is software. Programs are "Free Software" as defined in the FSD as long as their source codes (most preferably together with the binaries) are made available under a license that allows not only to use the software, but also do develop and distribute it. Any restrictions on standards used by a program do not affect compliance with the FSD at all.

Also, we may conclude that for a software license to be FSD-compliant it has to allow users for all three activities covered by copyright (copying, modification, distribution). The license cannot impose any material restrictions, in particular contract restrictions. The FSD does not contain any exhaustive list of forbidden restrictions, but it specifically forbids to condition the freedoms upon (1) payment, (2) notification, and (3) compliance with export control regulations. Nevertheless, it is allowed to impose some minor conditions specified in the optional requirements of the FSD. We identify them as: (1) distribution rules, (2) packaging rules, (3) publishing rules, and (4) reporting rules. These conditions are of minor importance, because the overall goal of the FSD and the practice of its application do not allow to use them to affect user freedoms materially.

Additionally, we may conclude that the addressees of the FSD are copyright holders (licensors) only. This means that an FSD-compliant license granted by the copyright holder in a program is sufficient for the program to become Free Software. Any restrictions applied by third parties, such as distributors or patent holders, escape the scrutiny. Thus, it is possible to call a program "Free Software" even though it would be subject to restrictions outside the FSD. In other words, compliance with the FSD means that the freedoms have been granted, but it does not mean that they are protected under all circumstances, and that they may be exercised in a particular context.

---

53 See FN 86.

54 Such a relation may be regulated by a Free Software license (*e.g.*, using *copyleft*), but there are licenses that do not contain such regulations (*e.g.*, *non-copyleft* licenses), which are still FSD-compliant.

In this thesis we follow the FSD and treat as "Free Software" any software that is subject to an FSD-compliant license. Subsequently, we analyse how the freedoms in such software are protected in a particular context of software communities and eGovernments. Therefore, we do not adopt another possible approach, which would be to treat software as "Free Software" only if users may exercise all their freedoms under any circumstance.

2.2 *Open Source Software*

Essentially, Open Source Software is available to any interested party together with the rights to control its working. The rights encompass source codes, not only the binaries. Thus, a question arises whether "Open Source Software" is the same as "Free Software". In order to answer this question and to explain the relationship between these two terms, it is necessary at least to analyse the formal Open Source Definition ("OSD").[55] The OSD is used by the Open Source Initiative to award certification marks for software licenses that fulfil ten minimum criteria for the scope of rights set thereto. These minimum criteria are as follows.[56]

(1) Free redistribution
(2) Access to source code
(3) The right to prepare and distribute derived works
(4) Protection of integrity of the author's source code
(5) No discrimination against persons or groups
(6) No discrimination against fields of endeavor
(7) Distribution of license as an adhesive standard form
(8) License must not be specific to a product
(9) License must not restrict other software that is distributed along
(10) License must be technology-neutral.

The FSD and the OSD cannot be easily compared. Certainly, it is possible to indicate the following relations. For example, "free redistribution" corresponds with Freedom 2, the freedom to *redistribute*. "Access to source codes" and "the right to prepare and distribute derived works" each correspond with both Freedom 1 (the freedoms to *study* and *adapt*) and Freedom 3 (the freedoms to *improve* and *release).* "No discrimination against persons or groups", "no discrimination against fields of endeavor", "license must not restrict other software that is distributed along", and "license must be technology-neutral" all correspond to some extent with Freedom 0 (the freedom

55 Open Source Initiative, *Open Source Definition*, at: http://opensource.org/docs/definition.php.

56 See: the *Open Source Definition*, *op.cit.*. The document elaborates on all these 10 criteria. For a good commentary on the OSD see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 8 *et seq*.

to *run* the program). The remaining conditions of the OSD do not correspond with the FSD in such a direct way. It follows that the OSD covers all requirements of the FSD and probably some additional requirements.

However, in practice the OSD has been applied in the evaluation of licenses in a similar manner as the FSD. All most popular model Free Software licenses are certified as "Open Source".[57] Thus, it seems reasonable to assume that there are no substantial normative differences between software referred to as "Free" and "Open Source".[58] Additionally, since this thesis is about the protection of user freedoms as defined in the FSD, there is no need to undertake any further analysis of the OSD.

2.3 *Open standards*

The notion of "open standards" is closely related to Free Software. We have highlighted this relation already when discussing the combinations between software and standards in Subsection 1.1.3. The essence is that a Free Software program may be required to interoperate with other programs. Such other programs may use a closed standard, or an open standard. Either way, the choice of a standard affects the capability of Free Software programs to interoperate with such other programs.[59] Consequently, the choice of a standard

57 Compare: Free Software Foundation, *Various Licenses and Comments About Them*, at: http://www.gnu.org/philosophy/license-list.html with: Open Source Initiative, *The Approved Licenses*, at: http://opensource.org/licenses/.

58 See: Richard M. Stallman, *Why "Free Software" is better than "Open Source"*, at: http://www.fsf.org/licensing/essays/free-software-for-freedom.html.

59 Interoperability is often defined as the capability of different elements to exchange information, to understand it and reuse. There are three levels of interoperability: (1) technical interoperability means the set of technical requirements that have to be observed in order to allow the exchange of information between IT systems; (2) semantic interoperability means the ability to understand the information exchanged; and (3) organizational interoperability, which is the ability to use and reuse the information across the systems (European Commission, IDABC, *Linking up Europe: the Importance of Interoperability for eGovernment Services, Commission Staff Working Paper*, 7 (EC 2003), at: http://europa.eu.int/idabc/servlets/Doc?id=1675). For example, in order for an application to interoperate properly with an operating system, it has to use the interfaces of the system. With the help of interfaces, it is also possible for one application to interoperate with another one. Network protocols provide for the interoperability of the layers of a network. (On "layers" see: Wikipedia, *OSI Model*, at: http://en.wikipedia.org/wiki/OSI_model) In case of the Internet, its interoperability is strongly dependent on the design of its standard protocols according to the layer transparency principle and the end-to-end principle. (On "layer transparency" see: Lawrence B. Solum, Minn Chung, *The Layers Principle: Internet Architecture and the Law*, 79 Notre Dame Law Review 815, 816 (2004); on "end-to-end" see: J.H Saltzer, D.P. Reed, D.D. Clark, *End-to-End Argument in System Design*, 2 ACM Transactions in Computer Systems 277 (1984), at: http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf). As far as the data formats are concerned, two given applications would not be capable of interoperation unless the sender's application provided the data in a format supported by the recipient's application.

by developers of other programs indirectly affects user freedoms in the Free Software program. Below, we explain this indirect relation between open standards and Free Software in more detail.

In this thesis we refer to standards of (1) interfaces, (2) protocols, and (3) data formats.[60] Interfaces may be defined as communication methods between all elements of a computer (*i.e.*, its hardware, operating system, and applications). Protocols usually refer to the methods of communication between programs (and other elements of IT systems) that interoperate over a network, such as the Internet. Data formats refer to the representation of the information constituting the communication itself. Thus, standards should not be mistaken for software. Rather, they only specify how software should work (precisely speaking, how it should interoperate). As we already explained, in practice interoperability depends on the access to "standard specifications". Specifications are documents that contain the information necessary to develop a program that is able to interoperate using the standard in question.[61]

For a standard to be open, its specification should not be just available, but also contain as much quality information as possible. The quality influences the degree of interoperability of programs and the number of different programs that may interoperate.[62] Additionally, there should be only minimum restrictions on the use of the information necessary to implement an open standard, and on the use of the implementations. It is possible to imagine multiple sources of such restrictions,[63] but the ones most commonly referred to in discussions about the openness of standards are *patents that are*

60 For a widely accepted definition of a "standard" in general see: ISO/IEC Guide 2:2004 *Standardization and related activities – General vocabulary*.

61 If the specification of a standard is ambiguous or incomplete, access to the standard's reference implementation may also be necessary. If a standard is implemented in software the access to its reference implementation would encompass making available its source codes. Conversely, if there is no specification available, the knowledge of source codes or decompiled binaries of a program that implements a standard, theoretically allows a sufficiently skilled person to enable a program to interoperate by using this standard. However, reconstructing interfaces, protocols, and data formats by studying binaries or even source codes, without access to specifications is extremely inefficient and impracticable.

62 Bruce Perens indicates that open standards must allow a wide range of implementations that may be created by various entities (businesses, academia, public projects). See: Bruce Perens, *Open Standards Principles and Practice*, at: http://perens.com/OpenStandards/Definition.html.

63 For example, the practice of "embrace-and-extend" is based on offering standard implementations together with extensions that themselves are not standard-compatible. See: Bruce Perens, *Open Standards Principles and Practice*, at: http://perens.com/OpenStandards/Definition.html.

material to the standard.[64] All these elements depend in practice (1) on designers of standards who prepare their specifications and (2) on holders of material patents. The designer and the patent holder may occasionally be the same person.

In this thesis, we would like to follow the definition of open standards included in the European Interoperability Framework v. 1.0 (henceforth the "EIF").[65] The EIF defines open standards as:[66]

(1) adopted and maintained by a not-for-profit organization, with its development process occurring on the basis of an open decision-making procedure available to all interested parties;
(2) published with the specification available for free or at a nominal charge, with the right to copy, distribute, and use it;
(3) the intellectual property of the standard made irrevocably available on a royalty-free basis; and
(4) without constraints on the re-use of the standard.

From the above we may conclude that the indirect relation between open standards and user freedoms is as follows. The less restrictions on a standard, the more programs may use it to interoperate. In particular, an open standard may be implemented in Free Software without any major problems. This allows to use such software in an environment consisting of other programs based on the open standard. Thus, it allows to make such software work better. Undoubtedly, this reinforces the freedoms, because it makes it more easy and practical to use, adapt, distribute, and improve programs, especially in a networked environment. Otherwise, these activities would be either impractical, or lead to an infringement of somebody's right (*e.g.*, a material patent). Therefore, it is necessary to analyse the notion of open

64 See, *e.g.*, Janice M. Mueller, *Patent Misuse Through the Capture of Industry Standards*, 17 Berkeley Technology Law Journal 623 (2002) (arguing that "patent owners should have a mandatory obligation to disclose the existence of any patents or pending applications that are material to the standard during their participation in the standards-setting process" (at 630)).

65 At the moment of this writing (2009), the consultations of the EIF 2.0 have finished. See: http://ec.europa.eu/idabc/en/document/7728. For a good overview of other understandings of the notion of "open standards" see: WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.

66 The European Interoperability Framework (EIF, at: http://europa.eu.int/idabc/servlets/Doc?id=19528), is the document that "represents the highest-ranking module of a comprehensive methodological tool kit for implementing pan-European eGovernment services" (*id*. at 5). EIF defines open standards at page 9. See: Nicos L. Tsilas, *The Threat to Innovation, Interoperability, and Government Procurement Options From Recently Proposed Definitions of "Open Standards"*, Special Issue Global Flow of Information, Autumn 2005 (at: http://www.ijclp.org/10_2005/pdf/ijclp_08_10_2005.pdf).

standards while discussing the protection of user freedoms. In this thesis we will undertake such an analysis in particular in the context of eGovernments (see Chapter 5).

2.4 *Software communities*

The beneficiaries of freedoms granted to Free Software may be individuals, firms, government agencies, *etc*. Many such users remain passive and exercise only Freedom 0, by using Free Software for a purpose of their choice. Other freedoms are exercised by active users only. Many such active users organize themselves in *communities*. In this thesis, we use the term *software community*, or simply *community*, to describe a group of entities that collaborate in the development of a Free Software project and that may also distribute the project to other users, as well as that provide guidance on its use.[67]

We use the word *project*, not *program*, in the description of software communities because not all software communities are organized around single Free Software programs (such as the Linux kernel community). There are many communities of gathered around combinations of programs organized in a functional whole (such as GNU/Linux distributions, desktop environments, office suites, *etc.*). Strictly speaking, we should refer to at least three types of software communities: (1) communities of single Free Software programs, (2) communities of combinations of programs, and (3) the overall "Free Software Community".

Because there are many Free Software programs and the differences between them are large, there are many Free Software communities, and the overall "Free Software Community" is not homogeneous. In our research we focus on communities of hierarchical and quite formalized structures. They may be communities that maintain single programs (such as the Linux kernel) or communities that maintain combinations of programs (such as of a GNU/Linux distribution). We anticipate that such communities would have the strongest possible impact on user freedoms. Additionally, such communities may have developed the most significant and advanced Free Software projects. So, we do not include in the analysis less organized communities and one-person projects, as we assume that they do not involve any major qualitative change in the way user freedoms are exercised or protected.

67 This definition may not be in compliance with the strict sociological meaning of "community". For some sociological analysis, see, *e.g.*: Margaret S. Elliot, Walt Scacchi, *Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture*, in: S. Koch (ed.), Free/Open Source Software Development (Idea Publishing, 2004), book chapter at: http://www.ics.uci.edu/~wscacchi/Papers/New/Elliot-Scacchi-BookChapter.pdf (referring to the concept of a community of practice, defined as "a group of people who share similar goals, interests, beliefs, and value systems in a common domain of recurring activity or work").

We also do not analyse the overall Free Software Community, since we do not find that it has developed any organization sufficient for our analysis, but in passing we mention how the overall community affects user freedoms. It follows that there are projects and communities not referred to in this thesis, to which our findings and conclusions may not apply.[68]

We discuss how software communities affect user freedoms in Chapter 4.

2.5 *eGovernment*

We use the following understanding of eGovernment. It is any communication directed to or originating from public administration that is carried out with the use of information technologies, in particular with the use of computer programs, usually interoperating over a network such as the Internet.[69] The notion of eGovernment should not be limited to internal communications of public administration only. Here, we would like to adopt a broader understanding and even focus on communications with the individuals, firms, and other users of government services.

The types of communications that we would like to include in eGovernment are:

(1) making available various public information, *e.g.*, on the WWW,
(2) making it possible or sometimes even making it mandatory to use information technologies in order to exercise rights or perform obligations *vis-à-vis* the state, and
(3) realizing the notion of *open society* by initiating public consultations on draft laws or on any other decisions that affect public interest.

As we already indicated in Chapter 1, the government significantly affects all other actors in the scene of software and standards. This is mostly due to the fact that the government generates significant demand impulses in the software market, namely the demand for software to be used in eGovernment. The government does that by exercising the *dominium*, *i.e.*, activities such as procurement and performance of contracts. The government can also interfere using the *imperium*, *i.e.*, the power of the executive to issue regulations that bind the actors or the audience in the scene of software or

68 See: Kevin Crowston, James Howison, *The social structure of free and open source software development*, Firstmonday, at: http://www.firstmonday.org/issues/issue10_2/crowston/index.html; Sandeep Krishnamurthy, *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*, Firstmonday, at: http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html.

69 Compare: EC Communication, *The role of eGovernment for Europe's future*, (COM(2003) 567, not published in OJ); See also: Wikipedia, *eGovernment*, at: http://en.wikipedia.org/wiki/eGovernment.

standards. The *dominium* and the *imperium* are exercised by the executive branch of the government. Additionally, the government can interfere on the scene of software or standards using its two remaining branches: the legislative and the judiciary. In this thesis we would like to focus on the government (the executive branch) acting within its *dominium*. Precisely speaking, we will closely analyse the impact on user freedoms of the *dominium* when exercised in order to introduce eGovernment. Only after we find that as a result of using the *dominium* user freedoms are affected negatively, or the *dominium* is unable to affect them positively, we will discuss using the *imperium*, as well as the legislative or the judiciary, in the construction of an improved framework.

By using the standards as a criterion, it is possible to enumerate two variants of eGovernment:

(1) eGovernment based on closed standards (henceforth called "Closed eGovernment");
(2) eGovernment based on open standards (henceforth called "Open eGovernment").

In Chapter 5 we introduce sub-variants of eGovernments and discuss how all of them affect user freedoms.

3 Regulatory framework of Free Software

In this chapter we reconstruct a model of the current regulatory framework of Free Software. The framework consists of (1) rules that regulate user freedoms (*i.e.*, rules for using, developing, and distributing Free Software) and (2) relations between the rules. Below, we recall our findings so far and then stipulate a working programme.

In Chapter 1 we identified two subscenes of a world where humans and computer programs cooperate. These are (1) the scene of software itself and (2) the scene of standards used by software. We concluded that there are two necessary conditions for a person to be able to control the working of a program. There should be: (1) access to source codes of the program, and (2) access to specifications of the standards used by the program to interoperate with other programs. We also concluded in Chapter 1 that the conditions of the protection of user freedoms are the same as the conditions of the control over the working of programs. Consequently, (1) access to source codes and (2) access to specifications of standards are both necessary for the protection of user freedoms. So, we should identify rules for both software (in particular source codes) and standards (in particular specifications of standards).

In Chapter 2 we stated that Free Software is software that is available to any interested party together with certain rights. The rights allow anyone to distribute an original Free Software program.[70] Also, anyone is allowed to develop an improvement[71] of such a program, and distribute improvements.[72] This means that a user can obtain either (1) an original Free Software program, or (2) an improvement of such a program. Additionally, this

70 In fact no-one usually distributes the original program *per se*. Rather, *copies* of the program are distributed, while the original is kept by the copyright holder. Here, we use the term "original" to indicate a *verbatim copy* of the original program. Sometimes, when speaking in the context of copyrights and proprietary software, "original" is also used to mean "authorized by the copyright holder", not necessarily verbatim. We do not use this meaning of "original" in this thesis, since Free Software licenses contain such an authorization and it would be quite hard to indicate a "non-original" Free Software in this meaning. Nevertheless, it is possible to talk about "official versions" of Free Software programs (see Section 4.2).

71 Certainly, anyone is also allowed to make the program worse. For practical reasons, we focus only on such manipulations with programs that result in improvements.

72 An improvement may be a modification of the program (a patch, an update, or an upgrade). An improvement may also consist of the program combined together with another program to make a new product. Here, we include improvements that are made by combining one's own material with the program, as well as by combining the program with a third party program. By "combining" we intend to cover many different software engineering techniques, such as linking, interaction through APIs, creating plug-ins, *etc*. Different combinations can have different legal consequences.

Towards_an_Improved_def.indd 59 23-02-2010 10:49:17

means that a developer or a distributor of Free Software can be either (1) its copyright holder, or (2) a third party. There are numerous combinations between who develops and distributes Free Software, and what kind of Free Software is developed and distributed. These combinations form various development and distribution chains. We illustrate example development and distribution chains in Figure 3.1.

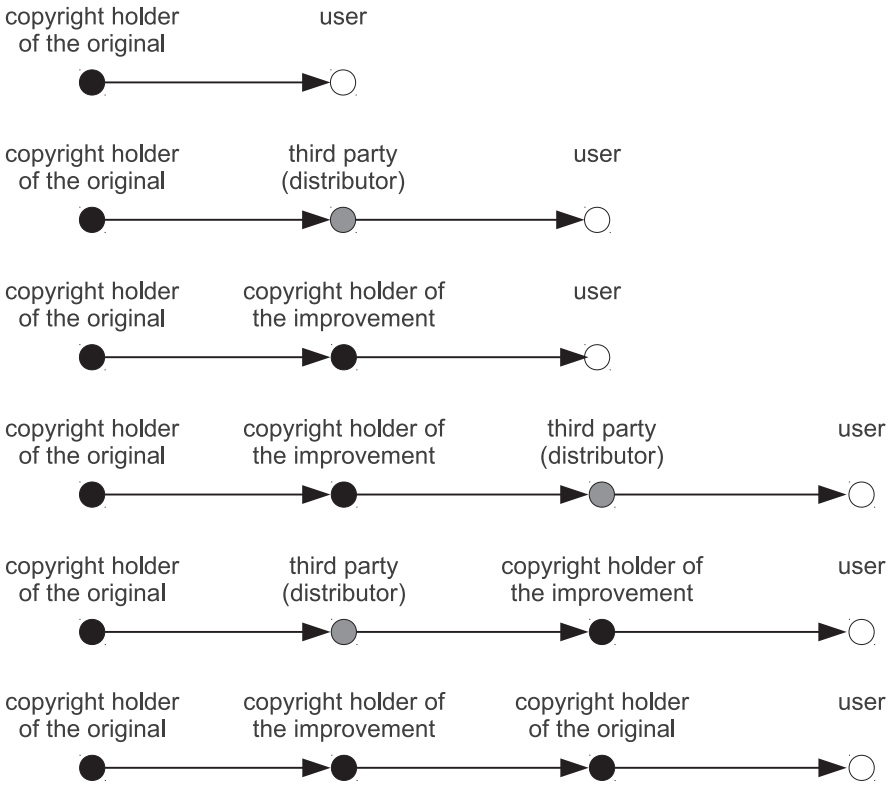

Figure 3.1: Example development and distribution chains

It is possible to imagine an unlimited number of additional development and distribution chains other than the ones presented in Figure 3.1. Yet, we need to construct a model of the framework applicable to all the chains. In order to do so, we have to group all of them into a limited number of classes. In Figure 3.1 we can see that certain chains are similar, at least from the point of view of the user (*i.e.*, at the last point of the chain). The similarity is that in each chain (1) the user obtains either an original or an improved Free Software program and (2) the user obtains them either from their copyright holders or from third parties. This means that any development and distribution chain can be assigned to one of four classes. There are four classes, because there are only four combinations between who distributes and what is distributed. We present these combinations in the following Table 3.1.

	copyright holder	third party
original	a user obtains an **original** Free Software program from the **copyright holder** of the original	a user obtains an **original** Free Software program from a **third party**
improvement	a user obtains an **improvement** of a Free Software program from the **copyright holder** of the improvement	a user obtains an **improvement** of a Free Software program from a **third party**

Table 3.1: Four combinations between who distributes and what is distributed

Some clarifications are necessary with regard to Table 3.1. The bottom row refers to improvements of Free Software. Given the unencumbered distribution of Free Software, there can be a case that someone distributes improvements copyrighted (developed) by someone else. Actually, this is a common situation in the Free Software scene. All these situations are represented in the bottom-right cell of Table 3.1. We remark that this cell covers both a situation when the improvement is distributed by a third party that has no copyright title to the improvement at all, as well as a situation when the improvement is distributed by the copyright holder of the original (who is then a third party from the point of view of the user and the copyright holder of the improvement).

Consequently, the bottom-left cell represents only a situation when the improvement is distributed by its copyright holder (the copyright holder of the improvement). As far as the top row is concerned, the top-left cell represents only a situation when the original is distributed by its copyright holder (copyright holder of the original). The top-right cell represents both a situation when the original is distributed by a third party without any copyright title, as well as a situation when the original is distributed by the copyright holder of an improvement of the original (who is a third party from the point of view of the user and the copyright holder of the original).

In each of the four combinations presented in Table 3.1 user freedoms could be affected. So, in this chapter we reconstruct a model of the current regulatory framework that includes rules (and relations between the rules) that regulate user freedoms in all these four combinations. Our process is as follows.

(1) We identify the rules for software and relations between them (see Section 3.1).
(2) We identify the rules for standards and relations between them (see Section 3.2).
(3) We reconstruct a model of the current framework for further analysis (see Section 3.3).
(4) Finally, we present chapter conclusions (see Section 3.4).

Below, whenever we include a rule or a relation in the model of the framework, we assign it with a number. For the sake of clarity we separately number rules and relations. "RUx" stands for a number of a rule, where x is a consecutive integer number. "REy" stands for a number of a relation, where y is a consecutive integer number.

3.1 Identification of rules for software and relations between them

We start the identification of rules for software by identifying *the default rule*. Then, we identify the remaining rules. *The default rule* can be identified by taking a closer look at the requirements of the FSD *vis-à-vis* the current copyright law. Essentially, the FSD requires to grant users their freedoms. More specifically, the FSD requires that all users of a program should be free to perform six activities with source codes and binaries of the program. These activities are: (1) running, (2) studying, (3) adapting, (4) redistributing, (5) improving, and (6) releasing. According to current copyright law, copyright holders have exclusive control over programs. Basically, they control the following three activities: (1) copying, (2) modification, and (3) distribution of programs. The exclusive control means that persons not authorized by copyright holders are not allowed to perform any of these three activities.[73] Notably, the six activities covered by the FSD are included in the three activities that may be restricted using copyright.[74] This means that by default copyright holders have exclusive control over the six activities covered in the FSD.[75] So, the freedoms do not exist as a default, and *the default rule* is as follows.

> ***The default rule***: Under the current copyright law, users are *allowed* to exercise all six activities covered by the FSD only if copyright holders authorize them to perform all three activities covered by copyright law. This means that copyright holders are *allowed* to prevent users from exercising all six activities covered by the FSD.
> In particular, no-one can become a user, developer, or a distributor of a program unless the copyright holder of the program agrees.

---

73 The exclusive control has certain exceptions, such as the idea/expression dichotomy, time bar, fair use, first sale, and no prohibition against independent creations of copyrighted works. Additional exceptions are provided for by some software-specific regulations, such as the limited decompilation right. These exceptions, however, do not affect the exclusive nature of control materially.

74 *Cf.* our findings in Subsection 2.1.2.

75 Nowadays, copyright protection applies automatically and without any formalities from the moment an original program is written. Also, owing to international law, such as the Berne Convention, TRIPS, and WIPO Copyright Treaty, copyrights have been significantly harmonized over the world. Copyright laws are imposed by governmental regulation. In the broadest sense, this rule is imposed and enforced by all government branches: the legislative, the executive, and the judiciary. Such regulation is generally binding. Thus, currently copyright holders easily obtain exclusive rights to software that are effective *erga omnes* in a significant number of jurisdictions all over the world.

A user can obtain an original Free Software program if only the copyright holder of the original agrees to make it available as Free Software. It follows that *the default rule* covers both combinations presented in the top row of Table 3.1 (*i.e.*, combinations related to original programs). The remaining two combinations (bottom row) are related to improvements. Improvements are usually *derivative works* or *collective works* based on the original program.[76] Under the current copyright law, the exercise of the freedoms in such improvements requires both (1) the consent of the copyright holder of the original, and (2) the consent of the copyright holder of the improvement. It follows that *the default rule* covers also both combinations presented in the bottom row of Table 3.1 (*i.e.*, combinations related to improvements). In all four combinations the respective copyright holders have to agree to the exercise of user freedoms. Thus, *the default rule* regulates all four combinations presented in Table 3.1.

We include *the default rule* as the first rule in the model of the framework (RU1). In the remainder of this section we identify other rules and relations between them that we also include in the framework. We first identify the rules and relations that follow from Free Software licenses (3.1.1). Then, we identify the other rules for software (3.1.2). Finally, we formulate our conclusions on rules for software and on relations between them (3.1.3).

3.1.1 Free Software licenses

Below, we analyse four issues related to Free Software licenses and we identify rules and relations that follow from these issues. The issues are: (1) *the grant of freedoms*, (2) *the right to fork*, (3) *copyleft*, and (4) *the hacker immunity*.

(1) The grant of freedoms

Consent of copyright holders is necessary for users to exercise their freedoms in all four combinations presented in Table 3.1. In case of an original Free Software program, the copyright holder of the original has to agree. In case of an improvement of a Free Software program, both the copyright holder of the original and the copyright holder of the improvement have to agree. As the reader may understand, copyright holders that follow the Free Software approach do agree that users exercise their freedoms. The usual form used

76 Theoretically, some improvements are neither derivative nor collective works based on the original program. In such a case they are either independent works of copyright, or they are do not qualify as copyrightable subject matter. Some copyrighted improvements can also constitute joint works. Here, we focus on such improvements that are copyrightable and are derivative or collective works based on the original program. For the sake of simplicity, we do not cover development of programs made in a direct collaboration of various copyright holders, which would lead to a creation of joint works. We address the joint work issue in more detail in Chapter 4.

by the copyright holders is a copyright license.[77] This means that software is coined Free Software by copyright holders themselves,[78] by licensing it in a way that conforms with all the requirements of the FSD. In Subsection 2.1.4 we concluded that any FSD-compliant license allows users to exercise all three activities covered by copyright (copying, modification, distribution). Every such license removes copyright restrictions and does not impose any additional material restrictions. This means that Free Software licenses contain a rule that can be expressed in the following way (we call this rule "*the grant of freedoms*").

> ***The grant of freedoms*** : Under a Free Software license, users (licensees) are *allowed* by the copyright holder to undertake all six activities covered by the FSD. This means that the copyright holder is *not allowed* to prevent users from undertaking these activities.
> In particular, any person (including hackers, firms, and governments) or group (*e.g.*, a software community) is *allowed* to use, develop, or distribute a Free Software program.

The relation between *the grant of freedoms* and *the default rule* is as follows. If a copyright holder,[79] who granted a Free Software license to a given original or improved Free Software program (the licensor), attempts to prevent a user, a developer, or a distributor (a licensee) from exercising the freedoms by invoking the exclusive copyright, the licensee may rely on the Free Software license as a legal defence.[80] Obviously, this is possible if only the license

77 Licenses are used also by copyright holders who follow the proprietary approach. However, proprietary licenses are used to maintain the exclusive control of copyright holders over their programs.

78 Such a method is usually referred to as "private ordering". See, *e.g.*, Yochai Benkler, *An Unhurried View of Private Ordering in Information Transactions*, 53 Vand. L. Rev. 2063, (2000); J.H. Reichman, Jonathan A. Franklin, *Privately Legislated Intellectual Property Rights: Reconciling Freedom of Contract with Public Good Uses of Information*, 147 U. Pa. L. Rev. 875 (1999); Amitai Aviram, *A Network Effects Analysis of Private Ordering*, (April 15, 2003), Berkeley Program in Law & Economics, Working Paper Series. Paper 80, http://repositories.cdlib.org/blewp/art8.

79 Third parties generally do not have standing as far as copyrights are concerned, *i.e.*, they cannot invoke *the default rule* unless they also hold a copyright title in the program.

80 A Free Software license does not exclude the possibility of invoking *the default rule* (copyright law) in case the licensee is in breach of the license. This issue is differently regulated in various laws. For example, under the U.S. law licensor can rely on copyright law in case of a license breach provided that the breached provisions are conditions not covenants (see: *Jacobsen v. Katzer* 535 F.3d 1373 (2008) and Lawrence Rosen, *Bad Facts Make Good Law: The* Jacobsen *Case and Open Source*, at: http://www.rosenlaw.com/BadFacts-MakeGoodLaw.pdf). Conversely, under Polish law it is generally believed that licensor cannot invoke exclusive copyrights against a breaching licensee, only contract-related claims. However, for uses outside of the license, copyright claims seem admissible. (See: the decision of the Polish Supreme Court of 20 May 1999, I CKN 1139/97, see also: J. Barta et al., Ustawa o prawie autorskim i prawach pokrewnych. Komentarz [Act on copyright and neighbouring rights. Commentary] (Dom Wydawniczy ABC 2001), commentary to art. 79).

is binding the licensor. If the license is not binding, *the default rule* is not affected and copyright holders can prevent the exercise of the freedoms. Free Software licenses attempt to bind automatically from the moment a person undertakes an activity covered by the freedoms. Namely, the consent for the exercise of the freedoms expressed in the licenses comes in a non-negotiable ("take-it-or-leave-it") form. If a person does not accept the license, the person cannot exercise the freedoms, since a consent of the copyright holder is a necessary condition of the freedoms.[81] Nevertheless, there is a legal question whether the license binds the user in all circumstances, even if it was not expressly negotiated and agreed upon. This definitely depends on the particular circumstances and the applicable law.[82] Here, we assume that most users as well as every developer and distributor of a Free Software program

81 A license is not the only possible legal basis for the mere use of a program. Various laws could provide alternative legal bases, such as fair use. For example, under Polish Copyright Act, Art. 75 (which implements the Software Directive Art. 5) a person who legally obtained a program can use it without a license, but only to the extent necessary to perform its purpose (unless contractually agreed otherwise). The provision does not in particular authorize to distribute the program, and it does not allow the person to access source codes of the program otherwise than after reverse engineering (which is further constrained in the law). Obviously, the exercise of the freedoms on the basis of Art. 75 alone is not possible.

82 Usually, Free Software licenses are included together with source codes (it has become a common practice to put a LICENSE or COPYING file in plain text together with files that contain the program proper). In case of binary packages, a license may be presented as a "click-wrap" but more often it is a part of the program's documentation or help file. For the validity of such a way to conclude a license under the U.S. law see, *e.g.*, *ProCD v. Zeidenberg* (86 F.3d 1447 (7th Cir. 1996) holding that a reference to a license outside of the box that contained software was sufficient to bind the user with the license, *M.A. Mortenson Co. v. Timberline Software Corp*. (970 P.2d 803 (Wash. Ct. App. 1999), aff'd, 998 P.2d 305 (Wash. 2000)) holding that agreement with license terms was sufficiently manifested by the installation and use of software by the buyer, even if it were not individually negotiated beforehand. A particularly interesting case in the Free Software context is *Microstar v. Formgen, Inc*. (942 F. Supp. 1312 (S.D. Cal. 1996), aff'd in part, rev'd in part on other grounds, 154 F.3d 1107 (9th Cir. 1998).) which concerned a level editor program of a popular computer game, Duke Nukem 3D. Neither reading nor accepting the license was the precondition of using the program. The license was contained in LICENSE.DOC file on the disk together with other program's files and was only referred to in the opening screen of the program. It was not presented on the computer screen, there was no notice on the box, and no fixed period to return and withdraw. The defendant argued that no average user would access the license file, but the court did not find this persuasive. For more U.S. caselaw analysis see: Kevin W. Grierson, *Enforceability of „Clickwrap" or „Shrinkwrap" Agreements Common in Computer Software, Hardware, and Internet Transactions*, 106 American Law Reports 5th 309. For case law related specifically to Free Software licensing see: *Wallace v. Free Software Foundation*, at: http://www.groklaw.net/pdf/WallaceFSFGrantingDismiss.pdf, *Harald Welte v. D. GmbH*, at: http://www.jbb.de/judgment_dc_frankfurt_gpl.pdf, or *Jacobsen v. Katzer* 535 F.3d 1373 (2008).

(original or improved) are *licensees* (while the copyright holder of the program is *the licensor*).[83]

The grant of freedoms in a Free Software license allows users to exercise their freedoms (1) only *vis-à-vis* the copyright holder who granted them and (2) only in the Free Software program subject to the grant (original or improved). First, under *the grant of freedoms* alone, third parties are not obliged to allow users to exercise their freedoms in a Free Software program.[84] So a distributor is allowed to distribute a Free Software program and restrict user freedoms at the same time. Second, *the grant of freedoms* in an original program does not by itself allow users to exercise any freedoms in improvements of the program. Certainly, users are allowed to create improvements on their own, but they are not allowed to exercise the freedoms in improvements created by others. It follows that *the grant of freedoms* covers all combinations presented in Table 3.1, but only to the extent that the consent of the copyright holder is necessary to exercise the freedoms. Such consent is indeed always a necessary, but not always a sufficient condition of the freedoms. We already noted that in case of improvements both the copyright holder of the original and the copyright holder of the improvement have to agree to the exercise of user freedoms. In the remainder of this thesis we will discuss many situations when the consent of copyright holders is not sufficient for user freedoms. For example, if the program (original or improved) is obtained from a third party (a distributor), the additional condition of user freedoms is that the third party does not restrict them.

We include in the model of the framework: *the grant of freedoms* (RU2), and the relation between *the grant of freedoms* and *the default rule* (RE1).

(2) The right to fork

Under *the grant of freedoms* licensees are in particular allowed to distribute a given original Free Software program. They are also allowed to develop an improvement of the program and distribute the improvement. Specifically, they are allowed to engage into such activities concurrently with other licensees (and even concurrently with copyright holders, the licensors). Concurrent development and distribution of the same Free Software program is called "forking".[85] From the point of view of user freedoms, forking is important in two situations: (1) when a licensee distributes an original Free Soft-

83 Certainly, the license can be non-binding in a particular case, or under a particular applicable law. Then, if there is no exception from *the default rule* provided for in the applicable law, the exercise of the freedoms constitutes copyright infringement (*the default rule* is not affected). Possibly, the user could use the software in a limited scope (see art. 5 and 6 of the Software Directive or their implementation in art. 75 of the Polish Copyright Act).

84 Notably, under the FSD a program is Free Software even if it is subject to restrictions imposed by third parties, such as distributors. *Cf.* our findings in Subsection 2.1.3.

85 See, *e.g.*, Wikipedia, *Fork (software development)*, at: http://en.wikipedia.org/wiki/Fork_(software_development).

ware program but restricts[86] users in the exercise of their freedoms, and (2) when the licensor or a licensee creates an improved Free Software program and does not distribute it as Free Software.[87] In the first situation any user can serve as a substitute source of the original program without restrictions, while in the second situation any user can create a substitute improvement and distribute it as Free Software. This is possible, because every Free Software licensee gives licensees *the right to fork*.[88] We identify the following rule in *the right to fork*.

> ***The right to fork*** : Any user of a Free Software program is *allowed* to distribute the original program or to develop an improvement of the program and distribute it (1) together with source codes and (2) without restricting the freedoms of other users.

The right to fork is in a relation with *the grant of freedoms*. Actually, it is a direct implication of *the grant of freedoms*. It is also in a relation with *the default rule*. The relation between *the right to fork* and *the default rule* is similar to the relation between *the grant of freedoms* and *the default rule*. Namely, if the copyright holder attempts to prevent a user from forking, the user can invoke the license (and *the right to fork* contained therein) as a legal defence.

In other words, *the right to fork* attempts to turn the rights granted to users in Free Software licenses into negative freedoms (liberties). Liberties are rules that prevent other parties from interfering with certain actions of a free individual. Under *the right to fork*, the licensor *allows* the licensees (1) to exercise all freedoms in the program or its improvements they created themselves. Licensees are also (2) *allowed to* distribute the program without restricting freedoms of other users as well as to grant freedoms to their improvements to other users. However, they are also (3) *allowed to* appropriate the program or the improvements. The licensees do not have any specific obligation to exercise their *right to fork*. So, *the right to fork* does not oblige

86 Many Free Software licenses contain clauses whereby the user that obtains the program from a licensee becomes an additional licensee *vis-à-vis* the original licensor. This means that distributors of Free Software are not expected to grant users a separate license (sublicense) wherein they could restrict user freedoms, but they "pass on" the initial license to users. Distributors are expected to grant licenses on their own only if they distribute improvements, of which they are copyright holders (granting of such licenses is required in *copyleft* clauses). H. Meekers calls this "direct licensing" (Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 29-30). See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 77. However, apart from "passing on" the original license, distributors can conclude contracts with users on their own, as well as sublicense the software (some Free Software licenses allow sublicensing). Such contracts can be used to restrict user freedoms.

87 Some authors present forking as negatively affecting software development due to the need to duplicate efforts. See: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 171.

88 See, *e.g.*, MeatBall, *RightToFork*, at: http://www.usemod.com/cgi-bin/mb.pl?RightToFork.

anyone to make the programs (or their improvements) available as Free Software. It follows that *the default rule* is affected if only someone decides to exercise *the right to fork*.

The rule identified in *the right to fork* covers all four combinations presented in Table 3.1. We remark that the coverage is not direct, since it does not allow users to access the specific original or improved programs that were not made available as Free Software. The rule merely allows users to become developers and distributors of substitute originals or improvements.[89]

We include in the model of the framework: *the right to fork* (RU3), the relation between *the right to fork* and *the default rule* (RE2), and the relation between *the right to fork* and *the grant of freedoms* (RE3).

(3) Copyleft

Under *the grant of freedoms* and *the right to fork* copyright holders (and everyone else, *ceteris paribus*) are not allowed to restrict user freedoms by invoking *the default rule*. But third parties are not required to refrain from doing so. They are also not obliged to ensure that users can exercise the freedoms. The source of such an obligation is another rule – the rule of *copyleft*.[90] *Copyleft*, briefly speaking, is an obligation of Free Software licensees (1) to allow users to exercise the freedoms with regard to an original Free Software program and (2) to grant users the freedoms to improvements of the program. They are required to perform in a particular way, not just refrain from certain activities.

If a licensee distributes an original or an improved Free Software program subject to *copyleft*, the licensee (the distributor) is obliged to make available the source codes of the program to such users to whom the program is distributed. Additionally, the licensee is not allowed to impose restrictions on the freedoms.[91] The rule that we identify in *copyleft* is as follows.

89 One could argue that users of proprietary software have *the right to fork* as well. Namely, they can independently develop and distribute substitute programs as well (this is mostly due to the fact that copyright law protects only expression, but not ideas embodied in the expression – for a landmark case see, *e.g.*, *Lotus Development Corporation v. Borland International, Inc.*, 516 U.S. 233 (1996)). However, they cannot do so using copyrighted material of a proprietary program. Conversely, Free Software licenses allow users to use such material when creating a substitute (a fork).

90 See: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 21, 72 *et seq*.

91 The distributor who is the copyright holder of an improvement of a *copylefted* program is obliged to license the improvement under a Free Software license. The *copyleft* clause in the Free Software license of the original usually indicates that the same model license should be granted to improvements. *I.e.*, the GPLv2 requires that the distributed improvements are released under the GPLv2, the MPL requires that they are released under the MPL, *etc.* Some Free Software licenses allow to distribute improvements under different licenses considered "compatible".

> ***Copyleft*** : The distributor of a Free Software program subject to a *copyleft* clause (original or improved) (1) is *obliged* to deliver the source codes of the program to users to whom the program is distributed and (2) is *not allowed* to restrict such users in the exercise of the freedoms in the program.

We remark that this rule applies only to such original or improved Free Software programs that are distributed.[92] Also, this rule does not apply to all imaginable distributed improvements, since it is generally based on the copyright holder's right to control derivative works (modifications),[93] which is included in *the default rule*.[94] Additionally, this rule applies to licensees only – Free Software licensors are not obliged to follow *copyleft*.[95] Namely, licensors are not obliged to deliver originals or improvements of their pro-

92 Distribution is the trigger of *copyleft* obligations under the GPL. However, *e.g.*, the *copyleft* clause of the Affero GPL covers improvements that are both distributed and used to offer services (such as by running them on a server, see: http://www.fsf.org/licensing/licenses/agpl.html). Similar trigger is included in the Open Software License and called "External Deployment" (http://www.opensource.org/licenses/osl-3.0.php). What constitutes "distribution" is usually defined in the applicable national law (see, *e.g.*, Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 234).

93 The exact relation between derivative works and modifications depends on the applicable law.

94 The exact scope of *copyleft* clauses varies depending on the license. The *copyleft* of GPLv2 (and GPLv3) covers all derivative works (for a thorough discussion about the exact scope of *copyleft* see, *e.g.*, Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7; Krzysztof Siewicz, *Scope of copyleft clause under Polish law*, J. Barta (ed.), Zagadnienia Prawa Autorskiego [*Copyright Law Issues*], ZNUJ PWiOWI [Jagiellonian University Intellectual Property Journal, Vol. 93 p. 235, (Zakamycze 2006), English version at: http://ksiewicz.net/files/siewicz_copyleft_scope.pdf. We are not aware of any Free Software license that attempts to cover all imaginable improvements, that is even the ones which are not derivative works (such as improvements that are only inspired by the original program), although the SleepyCat license could be considered an example of such an attempt (it covers "any accompanying software that uses the [licensed] software" (http://www.opensource.org/licenses/sleepycat.php). On the "10-line exception" in the *copyleft* clause of the LGPL see, *e.g.*, Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 40. On an insightful analysis of the GPLv2 *copyleft* clause and its scope see *id.* 200 *et. seq*. See also: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004) (discussing various Free Software licenses).

95 Here, we assume that every user, developer, and distributor of Free Software becomes a licensee, except for the copyright holder (who is the licensor). If a person distributes own improvements of a Free Software program, such a person most likely is both a licensee (towards the copyright holder of the original) and a licensor (towards the recipients of the improvement). Some Free Software licenses contain obligations of the licensors to provide source code of software, but this is not a part of the *copyleft* as discussed here (see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 25).

grams as Free Software, unless they want to, but they cannot prevent users from forking.[96]

The relation between *copyleft* and *the default rule* is as follows. *Copyleft* as such is based on *the default rule*, since without it the licensor could not control the use, development, and distribution of a program and its improvements. *The default rule* provides for a firm legal basis for imposing obligations on licensees whenever they undertake activities covered by exclusive copyrights. *Copyleft* attempts to turn the rights granted to users in Free Software licenses into positive freedoms. Positive freedoms are rules that oblige certain parties to allow free individuals to exercise these freedoms. Namely, *copyleft* is an *obligation* of the licensees to allow users to exercise their freedoms in the program or its improvements distributed by the licensees. *Copyleft* attempts to provide an enforceable claim against a person that would like to make a Free Software program or its improvement proprietary. Thus, *copyleft* attempts to *allow* to enforce the obligation of the licensees.

The relation between *copyleft* and *the grant of freedoms* is similar to the relation between *the right to fork* and *the grant of freedoms*. However, while *the right to fork* is a direct implication of *the grant of freedoms*, *copyleft* is a rule independent of *the grant of freedoms*. It can be included in a Free Software license together with *the grant of freedoms*, but there are Free Software licenses without *copyleft* (so called "academic" or "permissive" licenses, such as the BSD license). In such licenses, *the grant of freedoms* and the resulting *right to fork* are the only protection of the freedoms.

The relation between *copyleft* and *the right to fork* is as follows. In such cases on which *copyleft* does not extend, users have to rely on *the right to fork* only. *The right to fork* applies to every licensee of Free Software. *Copyleft* obligation applies only to licensees of such Free Software programs that are released under a Free Software license that contains a *copyleft* clause.

The rule identified in *copyleft* covers only three combinations in Table 3.1. The combination in the top-left cell is not covered. Namely, the copyright holder of the original who grants a Free Software license containing a *copyleft* clause does not assume an additional obligation not to restrict user freedoms, apart from the obligation already included in *the grant of freedoms*. The identified coverage by *copyleft* of both combinations relating to improvements (the bottom row) is partial, since it does not cover all imaginable improvements in all imaginable circumstances. Namely, the exact scope of *copyleft* depends on license wording and the applicable law.

96 Precisely speaking, copyright holders are not allowed to prevent users from forking (*i.e.*, from exercising the freedoms in the original Free Software) because they continue to be bound by the Free Software licenses they granted themselves (*the grant of freedoms*). What they could do, however, is stop making the program (and its source codes) available at all. Additionally, if copyright holders distribute improvements of which they do not hold copyrights, and which are subject to *copyleft* clause imposed by their copyright holders, they themselves become the licensees bound by *copyleft*.

Towards_an_Improved_def.indd 70 23-02-2010 10:49:19

We include in the model of the framework: *copyleft* (RU4), the relation between *copyleft* and *the default rule* (RE4), the relation between *copyleft* and *the grant of freedoms* (RE5), and the relation between *copyleft* and *the right to fork* (RE6).

(4) The hacker immunity

All programs (including proprietary software) usually have bugs, do not have all necessary features, and are not able to interoperate properly with other programs. *Ceteris paribus*, this means that there is a high probability that any program will be defective in the light of law (non-merchantable or not fit for a particular user's purpose). This also means that the use of the program could result in damages. Additionally, given the increasing ubiquity of exclusive rights in intangibles (so-called "intellectual property rights"), the use, development, or distribution of a program can constitute an infringement of third party rights.

In all these cases the law protects the injured party and there is generally no immunity for persons to whom the liability for the injury is attributable, unless some special exceptions apply. According to the law, software transactions are usually subject to warranty of merchantability, warranty of fitness for particular purpose, and warranty of title by default.[97] Depending on circumstances, warranty obligations can attach to a developer or a distributor of a Free Software program.[98] Liability for infringement of third party rights, depending on the nature of the infringement could be borne by developers, distributors, and also by users. If a user is held liable,[99] the user could have a recourse against the person who delivered the infringing software or otherwise contributed to the infringement. So, generally developers and distributors of Free Software could be liable for the software unless there is a rule to the contrary. The liability can also be attributed to users themselves *vis-à-vis* third parties, with a possible recourse against distributors or developers.

However, there is a common rule throughout the whole software industry that attempts to remove the liability from developers and distributors, and grant them with an immunity. As a result the risk is pushed onto the users, without leaving them with a recourse. Such an immunity has been embodied in various warranty disclaimers and liability limitation clauses. These clauses have been also widely accepted because many believe that it is technically impossible to write flawless software (both technically and legally), so actors have to "immunize" themselves from liability. The rule that follows from the warranty disclaimers and liability limitation clauses reads

97 For a general overview under the U.S. law see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 11.

98 Free Software can be developed and distributed both by its copyright holders as well as by third parties.

99 For example, not just the development and distribution, but a mere use of a patented invention without a valid patent license usually constitutes patent infringement. See: *Aro Mfg. Co. v. Convertible Top Replacement Co.* 377 U.S. 476, 84 S.Ct. 1526 (U.S.Mass. 1964).

that users cannot claim damages. The Free Software approach has eagerly borrowed this rule from the general regulatory framework of software.[100] Thus, the broad grant of rights in Free Software licenses is usually followed by an "as is" or other type of warranty disclaimer and liability limitation clause. This means that according to Free Software licenses there is a rule, which we refer to as "*the hacker immunity*".

The hacker immunity: Users are *not allowed* to claim liability related to Free Software.

We call this rule "*the hacker immunity*" because it removes a strong demotivator (liability) away from many volunteer hackers who are developers of Free Software programs. However, precisely speaking it removes liability from all Free Software licensors. Depending on a particular case the licensor can be a hacker, a firm, or any other entity that is the copyright holder of a given Free Software program (original or improved) and offers it under a Free Software license.

Two clarifications are necessary:

(1) *The hacker immunity* as contained in Free Software licenses does not *per se* remove liability from a person that is not the licensor. This means that such developers and distributors who are not licensors bear liability for the program by default. Usually, however, third party distributors accompany the distributed program with a separate warranty disclaimer and liability limitation clause and thus they extend the *hacker immunity* on themselves.[101]

(2) All Free Software licenses known to us contain *the hacker immunity* that covers technical defects (*i.e.*, they exclude warranty of fitness for purpose, warranty of merchantability). Many of the licenses extend the immunity on legal defects as well (*i.e.*, they waive liability for infringement of third party rights, the so-called "warranty of title").[102]

100 Some argue that, on the contrary, "the lack of warranty protection for licensees is a defining characteristic of [the Free Software] licensing model" and that proprietary licensors offer warranties and indemnifications (Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7, 85). However, most so-called end-user licenses of proprietary software contain as broad warranty disclaimers and liability limitations as are usually found in Free Software licenses, but without giving end-users access to source codes or any significant rights to software.

101 Notably, licenses such as the GPLv2 require that distributors of the program accompany it with the liability limitation clause in a prescribed form. The distributor, however, can choose to offer additional warranty or services. If such a warranty or services are offered, *the hacker immunity* continues to apply between users and the licensor, but users can benefit from warranty services of the distributor.

102 For an example of a license that does not release the licensor from all liability for legal defects see, *e.g.*, *Eclipse Public License* that contains representation that the contributor has sufficient copyrights to grant license (http://www.eclipse.org/org/documents/epl-v10.php). See also clauses 3.4.a and 3.4.c of the MPL.

There is no direct relation between *the hacker immunity* and other rules in the framework identified so far. Rather, *the hacker immunity*, by removing the demotivating liability out of the way of actors in the Free Software scene serves as a basic incentive mechanism that stimulates many actors not to abandon Free Software development or distribution. In particular, it stimulates them not to invoke *the default rule* against users but to grant them their freedoms. So, there is an indirect relation between *the hacker immunity* and *the default rule*.

The hacker immunity covers all four combinations presented in Table 3.1. Namely, it attempts to remove liability regardless of who distributes Free Software and what kind of Free Software is distributed. The coverage, however, is partial, since only some Free Software licenses waive all imaginable liability (including the title warranty).

We include in the model of the framework: *the hacker immunity* (RU5) and the indirect relation between this rule and *the default rule* (RE7).

3.1.2 Other rules for software

The default rule makes the consent of copyright holders a prerequisite of the freedoms. Copyright holders who follow the Free Software approach use Free Software licenses to grant users the freedoms (*the grant of the freedoms*).[103] The licenses also contain rules that attempt to protect the freedoms of users, *i.e.*, *the right to fork* and *copyleft*. Additionally, *the hacker immunity* attempts to release actors in the Free Software scene from liability. But the model of the framework including all these rules and relations between them (as identified in the previous subsection) is not complete. *The grant of freedoms* in Free Software licenses, and the protection of the freedoms by *the right to fork* and *copyleft*, as well as by *the hacker immunity* depend on whether these rules are enforceable and remain effective despite other rules that exist in the framework. Some of the other rules may dilute but some of them may reinforce these rules. Either way, the other rules can affect the freedoms in some or all

103 Some users exercise their freedoms in a software community, while other users exercise these freedoms in private or otherwise outside of the communities. Some of the Free Software users are governments or users of eGovernment services. Here, we may already assume that the communities are in a relation to user freedoms. We may also assume that eGovernments are also in a relation to user freedoms. However, we do not include these relations in the model of the current framework. Otherwise, we would not be able to answer RQ1 and RQ2, which are: "In what way do software communities affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?" and "In what way do eGovernments affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?" We shall include in the model of the framework only the rules (and relations between them) that regulate the freedoms independently of them being exercised in the context of communities or eGovernments. Then, (in Chapters 4 and 5, respectively) we will analyse how such a model of the framework is affected by software communities and eGovernments.

combinations presented in Table 3.1. We need to include these other rules in the model of the framework.

Below, we identify the remaining rules for software and their relations with the rules identified in Free Software licenses. In order to identify these rules we focus on the following issues: (1) license proliferation and incompatibilities, (2) license revocability, (3) *inter partes* nature of licenses, (4) software-related patents, (5) contracts with distributors, (6) liability rules, and (7) non-legal regulators of software. At the end of each point we state what rules and relations we include in the framework.

(1) License proliferation and incompatibilities

Since mid 1980s, few dozen of FSD-compliant model licenses have been prepared.[104] By far the most popular of them is the GNU General Public License version 2 ("GPLv2" or "GPL"), which applies to approximately 70% of Free Software.[105] The GPLv2 and the second most popular model license (GNU Lesser General Public License, "LGPL") have been prepared by the FSF. The FSF has recently (2007) finalized a public debate on the "upgrade" of the GPLv2 and LGPL. As a result, the new version 3.0 of the GPL ("GPLv3") and the new version 3.0 of the LGPL ("LGPLv3") have been adopted. Not all licensors that have been using GPLv2 have switched to GPLv3, although a growing number has switched already.[106] Other popular model licenses include, for example, Berkeley Software Distribution License (the "BSD license"), and Mozilla Public License (the "MPL"). We illustrate the proliferation in Figure 3.2.

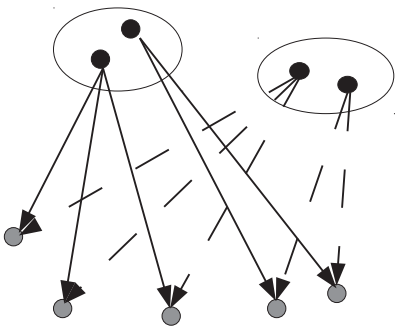

Figure 3.2: Proliferation of licenses

104 Free Software Foundation, *Various Licenses and Comments About Them*, at: http://www.gnu.org/philosophy/license-list.html; Open Source Initiative, *The Approved Licenses*, at: http://opensource.org/licenses/.

105 Free Software Foundation, *GNU General Public License*, at: http://www.gnu.org/licenses/gpl.html, the popularity of the GPL is based on data available from SourceForge repository (http://sourceforge.net).

106 Many programs are released under "GPLv2 or any later version", which seems to allow users to choose that GPLv3 applies to them. The effectiveness of such licensing depends on the applicable law.

In Figure 3.2 we present a result of using two different model licenses. Some licensors (black dots demarcated in one circle) use one model Free Software license, while other licensors (the other black dots demarcated in the other circle) use another model Free Software license. Each user (grey dots) obtains one program under the first license and one program under the second license. As a result, the rights and obligations of each user towards each licensor are subject to a different license (this is represented by different lines between users and licensors). It follows that if a user obtains more than one Free Software program (original or improved) and the obtained programs are subject to different Free Software licenses, the freedoms of that user should be scrutinized with regard to each program separately. This is often the case in practice, since even simple IT systems consist of numerous programs. Thus, we should expand the four combinations presented in Table 3.1 by adding one additional table of four combinations for each program subject to a different Free Software license that a licensee uses. We illustrate this in Figure 3.3.

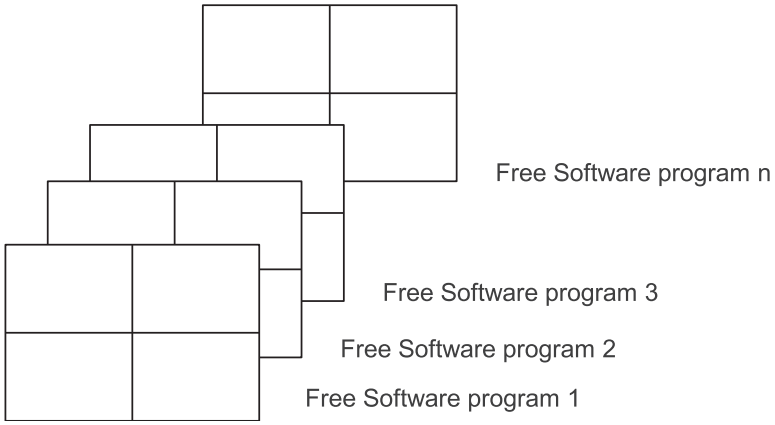

Figure 3.3: Using programs under different Free Software licenses

It would be impractical to reconstruct a model of the framework with a degree of detail that is necessary to account for all different Free Software licenses. In such a case, we would have to reconstruct a separate framework for each license and then a combined framework for combinations of programs under different licenses. It follows that in order to make our task practicable, we have to downsize the model to be able to deal with the issue of proliferation on a more general level. We do the downsizing below, by identifying rules that are materially affected by the proliferation. Then, we include in the model these rules as adjusted to account for the proliferation to the extent it materially affects them.

As far as *the grant of freedoms* is concerned, since the FSD requires to allow users to exercise all six activities enumerated thereto, all FSD-compliant licenses allow licensees to exercise these activities. Also, a given Free Soft-

ware program generally remains subject to one Free Software license, not to many different licenses at the same time. This means that *the grant of freedoms* is not affected materially, as long as users obtain single original Free Software programs (regardless of whom).[107] However, in practice distributors usually combine multiple Free Software programs together and offer them to users as complex software products.[108] The combination can be limited to simple collection of programs in a single distribution, but usually it involves at least some pre-configuration of them to work together properly. It is often the case that programs are combined using various software engineering techniques, which goes beyond mere aggregation of them. Certainly, a distributor of any combination has to comply with all licenses of the combined programs at the same time. This is possible if only all these licenses are *compatible.*

Licenses are compatible with each other as long as they permit to combine programs released under their terms and to distribute the combination.[109] Distribution of combination is possible if only all licenses of the combined programs are complied with. If one license obliges the licensee to perform in a particular way, while another license forbids the licensee to perform in such a way, then the licensee cannot comply with both of them at the same time. For example, if two licenses contain *copyleft* clauses that require improvements to be distributed under different Free Software licenses, such two licenses are incompatible. Licenses are incompatible also if only one of them contains a *copyleft* clause, while the other imposes on the user other obligations, which cannot be performed together with distributing improvements under the Free Software license required by the former.

But incompatibilities occur not only between some Free Software licenses. Naturally, the Free Software licenses are to a significant extent incompatible with the proprietary software approach. Developers and distributors of proprietary software usually cannot use Free Software in their programs and

107 Certainly, users have to scrutinize each license separately, in order to determine the exact scope of the rights and obligations. Although we can assume that all FSD-compliant licenses grant users all their freedoms, some of them are written better than others, and the differences may lead to their different interpretations. See *e.g.*, Brendan Scott, *BSD – The Dark Horse of Open Source*, at: http://opensourcelaw.biz/publications/papers/BScott_BSD_The_Dark_Horse_of_Open_Source_070112lowres.pdf (arguing that the BSD license requires BSD code and modifications to BSD code to be licensed under the terms of the BSD license, while the BSD license is commonly believed not to include a *copyleft* clause). We will return to the issue of different interpretations in Chapter 4.

108 For example, modern GNU/Linux distributions include hundreds of Free Software programs. A user could attempt to obtain all these programs directly from their copyright holders, but this is rarely the case given the time and effort necessary to find and configure all of them to work together.

109 According to the FSF, compatibility with the GPL means that "*you can combine code released under the other license with code released under the GNU GPL in one larger program.*" (See: http://www.fsf.org/licensing/licenses/gpl-faq.html#WhatDoesCompatMean). On license incompatibilities generally see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 159 *et. seq*.

maintain exclusive control over them at the same time. Precisely speaking, *non-copyleft* ("academic" or "permissive") licenses are rarely a burden for proprietary actors, and many programs subject to such licenses have been included in proprietary software. Conversely, *copyleft* licenses do not allow appropriation of Free Software, unless the proprietary actor finds a way to avoid *copyleft* obligations. Certainly, this may lead to less Free Software being included in proprietary software, but it also means that it is harder to restrict user freedoms (which is the exact purpose of the *copyleft*). In other words, we do not consider that inability to appropriate some Free Software (*e.g.*, to include it as a part of a proprietary offering) is a limitation of user freedoms. Actually, we consider any such successful appropriation as a restriction of user freedoms, which should be prevented in order for the freedoms to be sufficiently protected.[110]

However, from the point of view of user freedoms such an inability to appropriate a Free Software program may constitute a problem in the following situation. Imagine that it is desired to use such a program in a certain conjunction with a proprietary program. This might be a case when the user is locked-in to a closed standard used in the proprietary program.[111] It may be possible to provide for interoperability by combining the Free Software program with certain proprietary extension, but the particular license of the Free Software program does not allow such a combination to be legally developed or distributed. Actually, many developers and distributors attempt to work around this issue using different means. The means usually involve restricting the freedoms of users or requiring them to trade the freedoms in exchange for the additional functionality.[112] But from the point of view of user freedoms, a proper resolution of the incompatibility with the proprietary licensing would be to provide users with a substitute of the proprietary software to which users are locked in, which would be completely free of restrictions. This can be done, however, only if the closed standard can be translated to an open standard, or if users are stimulated to switch to open standards otherwise. This is not an easy task, as we show later in this chapter.

Here, for the sake of simplicity, when discussing license incompatibilities we focus on incompatibilities between Free Software licenses only. We leave incompatibilities with the proprietary approach aside. We illustrate the incompatibilities between Free Software licenses in Figure 3.4.

110 See below on contracts with distributors and non-legal regulators (in particular passages about making Free Software part of a service or a device).

111 For the discussion about closed standards and lock-ins see Subsection 3.2.1.

112 The means include dual licensing, maintaining separate repositories with proprietary extensions, avoiding using software subject to *copyleft* licenses, or trying to circumvent *copyleft* clauses, *etc*.

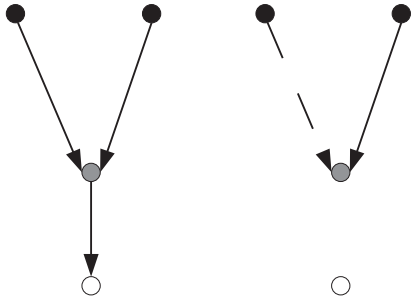

Figure 3.4: Incompatibility of licenses

Assume there are two Free Software programs, each licensed by a different licensor. For example, they may be two libraries[113] each able to expand a third program with new functionalities. On the left of Figure 3.4, we see that the licensors (black dots) use two compatible licenses for their programs (libraries). They grant the licenses to a user (grey dot). The user can exercise the freedoms in each of these programs separately. Additionally, the user is legally allowed to combine the programs together, and to distribute the combination. In such a way, the user creates a complete Free Software product and offers it to yet another user (white dot). It is possible since both licenses for the "input" programs are compatible and allow to redistribute them as combined in a finished product under one license. Conversely, on the right of Figure 3.4, we see that the licensors use two incompatible licenses for their programs. If both licenses are Free Software licenses, the user may exercise the freedoms in the two programs. In particular, the user is allowed to improve the programs separately and separately distribute the improvements. However, the user cannot satisfy obligations in both licenses and distribute the programs combined into one product.[114]

Combination may constitute a collection of multiple original Free Software programs but it may also be an example of an improvement of these programs.[115] Depending on the exact wording of a particular license, results

113 For an explanation of "library" see: Wikipedia, *Program library*, http://en.wikipedia.org/wiki/Program_library.

114 An interesting example is the the SleepyCat license (http://www.opensource.org/licenses/sleepycat.php, pointed out by Heather Meeker (Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 43.), which allows to distribute improvements under another Free Software license (the relevant part reads: "The source code must either be included in the distribution or be available for no more than the cost of distribution plus a nominal fee, and must be freely redistributable under reasonable conditions").

115 Here, we use the word "combination" to cover various copyright institutes, such as "modification", "collection", "derivative work", "collective work". Each of these institutes has different legal consequences. When creating a particular contribution a user has to determine which of these institutes applies exactly and what are the consequences under the applicable law.

of incompatibilities between licenses can vary from a restriction on distributing any imaginable combination to a restriction on distributing specific combinations.[116] In practice, the proliferation and incompatibilities have been found to affect specific combinations only. Also, at least three types of tendencies can be observed that attempt to overcome the proliferation and incompatibilities. First, there is already a clearly visible trend in the Free Software licensing. Namely, more popular model licenses are chosen more often than the less popular, and rarely licenses are drafted anew.[117] Second, many licensors use dual licensing schemes, which can eliminate most if not all issues related to the proliferation and incompatibilities.[118] Third, the drafters of some most popular model licenses have already started releasing new versions of these licenses which contain clauses that address many incompatibilities.[119]

It follows that proliferation, and the resulting incompatibilities between Free Software licenses, will continue to affect *the grant of freedoms* in all combinations presented in Table 3.1 materially, unless something prevents the above tendencies to succeed. Currently (2009) we are still in a transitional phase, and it cannot be taken for granted that the proliferation and incompatibilities will be successfully eliminated owing to these tendencies alone. So, for the time being we consider that *the grant of freedoms* is affected, although only to the extent that (a) the combination includes Free Software programs released under incompatible licenses, and only to the extent that

116 For example, *copyleft* obligations under GPLv2 do not apply in case a "mere aggregation" of programs is distributed. So, combinations of programs under GPLv2 can be distributed with programs released under licenses incompatible with the GPLv2 as long as the combination is a "mere aggregation" only. What constitutes a "mere aggregation" is unclear from the legal point of view. However, the overall Free Software community has developed some customs and trade practices that provide some guidance. See, *e.g.*, Software Freedom Law Center, *Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers*, at: http://softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html.

117 Recently, the OSI has been rather reluctant to approve any new licenses as "Open Source", even if they would meet the OSD criteria. But there have been some new model licenses (MSPL, EUPL) approved by the OSI despite this reluctance.

118 Dual licensing means that a program is released under a number of model FSD-compliant licenses. Users (in particular users that combine the program with other programs) can choose which license they will be bound to. If such a choice is made possible by the licensor, it can be possible to choose a license compatible with the licenses of other combined programs. Arguably, after the choice there is no possibility to change it again when a subsequent combination is made. See: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 129.

119 See, for example, GPLv3, EUPL. See also: CeCILL v2 (containing an express GPL compatibility clause).

(b) such a combination is distributed.[120] Otherwise, *the grant of freedoms* is not materially affected. Strictly speaking, *the grant of freedoms* is limited by the following rule that we call "license proliferation and incompatibilities".

> **License proliferation and incompatibilities**: Users are *allowed* to distribute combinations of Free Software programs as long as licenses of all combined programs are complied with.

License proliferation and incompatibilities affect also the rules that we identified in *the right to fork* and *copyleft*. Namely, they limit the exercise of *the right to fork* with regard to combinations of programs. They also limit the exercise of *copyleft*. Here, one clarification is necessary. The incompatibilities do not release the distributors from their *copyleft* obligation not to restrict user freedoms. Rather, they make it legally impossible to observe such an obligation if the distributor attempted to distribute the combination of two or more programs that are subject to incompatible licenses. In such a case, the distributor has to refrain from distribution and can only use the combination in private. Certainly, other users are free to make and use the combination by themselves privately as well.[121]

All Free Software licenses known to us contain broad liability limitation clauses. When discussing *the hacker immunity* we noted that the clauses are sometimes different to the extent some of them extend to legal defects (title warranty), while others do not. This means that the actual liability may vary depending on the wording of the license in question. We do not consider this as a material difference, since the main purpose of *the hacker immunity* is to waive liability for technical defects (warranty of merchantability). Certainly, some Free Software licenses may boast of well-written waivers, while others can be found lacking some important wording. We do not find that this is the case with the most popular Free Software licenses *per se*; it depends more on the differences between national laws, under which such clauses would be scrutinized. So, we assume at this point that the proliferation does not affect materially *the hacker immunity*.

We include in the model of the framework:

(1) license proliferation and incompatibilities (RU6);
(2) the relation between license proliferation and incompatibilities, and *the grant of freedoms* (RE8);

120 Users are still allowed to combine most Free Software and use the combinations in private without distributing them, even if the combined programs are released under incompatible licenses. This is because incompatibilities usually occur between *copyleft* clauses (or *copyleft* clauses and other clauses imposing distribution-related obligations), which are usually triggered upon distribution only.

121 In the Free Software scene there exist many repositories of software that cannot be distributed together with another software, but can be legally used by end-users. See, *e.g.*, http://livna.org.

(3) the relation between license proliferation and incompatibilities, and *the right to fork* (RE9); and
(4) the relation between license proliferation and incompatibilities, and *copyleft* (RE10).

(2) License revocability

In Subsection 2.1.2 we found that the FSD requires irrevocability of the license as a condition of compliance. We can assume that a license that expressly provides for a right of the licensor to terminate it unilaterally (without a valid reason) would not be accepted as FSD-compliant by the FSF. So far, the FSD allows to terminate licenses only if the licensee "does something wrong". We are not aware of any Free Software licenses that are revocable as far as the license wording is concerned (*i.e.*, they contain no termination clauses except in case of a breach). However, it may be the case that the applicable law provides for a non-waivable right of the licensor to terminate a license, or it requires a special wording, or form, in order to make a license irrevocable by the licensor.[122] In such a case the copyright holder could revoke the license or otherwise terminate it unilaterally, despite the license wording. After the license has been terminated, the copyright holder could invoke *the default rule* against users and as a result prevent them from exercising the freedoms. Consequently, *the grant of freedoms* would be annulled, since it requires a binding Free Software license to exist.

A main question that arises reads: is such a license termination possible in practice? The answer is positive if the licensees are not numerous and if the licensor can easily identify all of them. In such a case, all licensees can be delivered with termination notices in a form as required by the applicable law. However, users usually obtain Free Software without identifying themselves, and attempts to identify them would require additional efforts that may prove impracticable. Even if it were possible to identify all users who obtained the program directly from the licensor, a significant part of Free Software distribution is performed by third parties. Free Software licenses

122 Under Polish law (Copyright Act) a license concluded for an unspecified period of time can be terminated by either party. It is questionable whether the licensor could waive the right to terminate. Conversely, a license granted for a specified period of time cannot be easily terminated unless parties agree otherwise. However, a license granted for a specific period longer than 5 years, after that period automatically transforms into a license granted for unspecified period of time, as a matter of law. For an American law analysis see: Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7, 41 (arguing that irrevocability can be obtained only if a Free Software license is concluded as a contract). See also: Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 228 (doubting whether Free Software licenses are "bare licenses" that can be revoked under the U.S. law). For a European law perspective see: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 51, 86 *et seq*.

do not require such third parties to report to whom they distribute. Also, termination of licenses of identified distributors does not necessarily terminate the licenses of downstream recipients.[123] It follows that in case of a significant user base or at least of a developed downstream distribution personal delivery of termination notices does not seem to be practicable. So, it has to be determined whether the licensor is legally allowed to terminate the license in another way, for example by declaring publicly that the license is revoked (terminated). This depends on the applicable law, and we believe it to be unlikely that every law would allow to terminate the license of all users in such a way. Nevertheless, the licensor could still manage to identify major users of the program (*e.g.*, its most prominent developers and distributors) and be able to serve them with termination notices. From the practical point of view, termination with regard to such major users might be sufficient to prevent most other users from exercising the freedoms. Thus, restriction of user freedoms using license revocability is possible in practice, even if it would not affect all users.[124]

We are not aware of a license to any major Free Software program having been revoked since the beginning of the history of Free Software licensing, which dates back to the 1980s. Actually, many licensors have even made public pledges to maintain their programs as Free Software.[125] But should the licensors suddenly change their mind, the revocation of their licenses would affect all four combinations presented in Table 3.1, since a valid Free Software license is necessary to exercise the freedoms in each of these combinations. So, we account for such a negative scenario of the revocability in the model of the framework by adding the following meta-rule. We call it "license revocability".

> **License revocability**: *The grant of freedoms* remains in force as long as the licensor does not effectively revoke the license.
> Revocation enables the copyright holder to invoke *the default rule* against users without leaving users with a legal defense.

123 Generally, a downstream user obtains automatically the license from the original licensor whenever the program is redistributed (see FN 86). Thus, assuming such mechanism is effective the revocation is possible if only all licensees may be quickly identified and delivered with termination notices, before the program is passed downstream.

124 Under particular applicable laws it may be possible to invoke additional legal arguments against termination. See, *e.g.*, Janusz Barta, Ryszard Markiewicz, Oprogramowanie Open Source w świetle prawa. Między własnością a wolnością [Open source software in the light of law. Between property and freedom] (Zakamycze 2005), 113 (arguing that under Polish law a revocation of a Free Software license is a misuse of rights ("nadużycie prawa") or is contrary to good morals ("zasady współżycia społecznego")).

125 See *e.g.*, Brian Prince, *Sun Asserts MySQL Will Remain Open Source*, at: http://www.eweek.com/c/a/Database/Sun-Asserts-MySQL-to-Remain-Open-Source/. Some pledges refer just to the price of software. See *e.g.*, Ubuntu, at: http://www.ubuntu.com/products/whatisubuntu ("Ubuntu is and always will be **free of charge**").

This rule has direct relations with *the grant of freedoms* and *the default rule*. A revoked Free Software license renders *the grant of freedoms* ineffective and it allows licensors to invoke *the default rule* against users.

We include in the model of the framework:

(1) license revocability (RU7),
(2) the relation between license revocability and *the grant of freedoms* (RE11), and
(3) the relation between license revocability and *the default rule* (*RE*12).

(3) Inter partes nature of licenses

Free Software licenses create *inter partes* rights and obligations between licensors and licensees. More precisely, this means that each license binds only the particular licensor *vis-à-vis* the licensee to whom the license was granted. It follows that the possibility to apply any rights and obligations from a Free Software license to persons other than its parties is significantly limited. The *inter partes* nature of licenses affects *copyleft*. Other rules in the framework are not materially affected. Essentially, *copyleft* obliges licensees to allow users to exercise their freedoms in Free Software programs (or their improvements) distributed by the licensees. However, as an *inter partes* obligation, it obliges the licensee only towards the licensor, not towards users (even if the obligation is to perform to the benefit of users). Generally, only a person to whom another person is obliged (here: the licensor) can legally enforce the obligation. Consequently, users (despite being the intended beneficiaries of *copyleft*) cannot enforce *copyleft* against licensees, unless there is a rule to the contrary.

Whether there is a rule that allows a user to enforce *copyleft* depends on the applicable law. There are laws that allow third parties to enforce certain *inter partes* obligations. For example, under Polish law it may be possible to construct *copyleft* clauses as "contracts to the benefit of a third party." If so, then a third party could have the standing necessary to enforce *copyleft* clauses without the help of the licensor.[126] Yet, there is a debate whether Free Soft-

126 According to art. 393.1 of the Polish Civil Code, if the contract obliges a party to perform to the benefit of a third party, such a third party may request performance directly from that party (unless this is excluded in the contract). The Polish Supreme Court held on 28 November 2003 r. (IV CK 206/2002) that the third party beneficiary should be identified in such a contract, or the contract should at least contain clauses that allow to identify the third party. Free Software licenses are model licenses that do not identify any particular third party as a beneficiary. However, licenses such as the GPLv2 are quite explicit that their *copyleft* clauses should benefit every user that receives the software from a licensee. Nevertheless, the GPLv2 at the same time provides that in case of a breach it is automatically terminated (Sec. 4). A terminated license does not bind parties any longer. This means that the licensee is no longer obliged to perform to the benefit of a third party. Also, art. 393.1 applies to contracts only, not to unilateral acts. It means that its applicability depends on the construction of Free Software licenses as contracts. For a similar analysis under U.S. law see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 154 *et. seq*.

ware licenses are contracts or rather unilateral acts.[127] For example, Raymond T. Nimmer argues that if *copyleft* is non-contractual in nature, a breach of *copyleft* could not be remedied by specific performance, although it could provide a basis for a copyright infringement suit.[128] If there are indeed legal limitations in demanding performance of *copyleft* obligations (be it by the licensor or a third party), then users can rely on *the right to fork* only.[129]

Here, we assume that *copyleft* obligations can be enforced, so we retain the *copyleft* rule in the framework. However, given the *inter partes* nature of the license relation we have to include an additional rule in the framework that constitutes a limitation of the *copyleft* rule. We call this rule "*inter partes* nature of licenses".

---

127 Licenses are based on copyright, and they additionally can be based on contract law. In common law jurisdictions, a license in this context is usually defined as a private unilateral grant of rights. Such a unilateral act is distinguishable from a contract, which is thereto defined as an agreement whereby one person promises something for an exchange of a consideration. In the U.S. such licenses that are unilateral acts are based on the federal copyright law only, while contracts are additionally regulated by the states. Many of the most prominent Free Software advocates, including Eben Moglen (with regard to the GPL), have often argued that Free Software licenses are unilateral acts (Eben Moglen, *The GPL Is a License, not a Contract*, at: http://lwn.net/Articles/61292/). But see: Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7, (arguing *inter alia*, that asking whether the GPL is a contract is nonsensical without a specific context in which a particular license incorporating the GPL's terms was concluded (at 25)). For the analysis of the GPLv2 *in abstracto* see Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 225 *et. seq*. See also: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 148 (discussing how even the language of most permissive Free Software licenses allows to find consideration necessary to establish a contractual relation in common law jurisdictions). Even the FSF does not exclude the possibility that software may be subject to contract law (see FSD). In civil law jurisdictions it is also possible to conclude that the use of programs may be regulated unilaterally by the copyright holder, although certain jurisdiction-specific reservations may apply. For example, under Polish law a license is generally a contract (although a specific one, subject to copyright law) and it is not clear whether a license could be granted unilaterally. See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 46, 55 *et seq*., 150 *et seq*.

128 Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7, 26. The author, however, provides an extensive argumentation for the contractual nature of Free Software licenses. See also Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 226-227.

129 Heather Meeker points out that if a program constitutes a joint work, all authors might be required to stand in a case. This might effectively prevent enforcement of *copyleft*, especially in projects, the rights of which are shared among multiple contributors. See: Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 179. See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 51, 72 *et seq*.

> ***Inter partes* nature of licenses**: Unless the applicable law provides to the contrary, only the Free Software licensors are *allowed* to enforce obligations of the licensees under *copyleft*.

Naturally, the above rule covers only those combinations in Table 3.1, which are covered by *copyleft* itself. Also, this rule is in a relationship with *copyleft* only. The relationship is that the obligations under *copyleft* are limited *inter partes*, unless there is a legal rule to the contrary in the applicable national law.

We include in the model of the framework the *inter partes* nature of licenses (RU8) and the relation between this rule and *copyleft* (RE13).

(4) Software-related patents

Under *the default rule* copyright holders are the ones who can restrict all six activities covered in the FSD. Other rules identified so far regulate the way in which copyright holders allow users to undertake these activities. However, copyright holders are not the only ones who can restrict users in the exercise of their freedoms. In other words, consent of copyright holders is always necessary, but not always sufficient for user freedoms. Notwithstanding copyrights, there are other exclusive rights that a third party might hold in relation to a Free Software program. The most important ones of such exclusive rights are patents.[130] Patent rights belong exclusively to their holder and they are effective *erga omnes*, including against the parties to a Free Software license.[131] With these rights, the patent holder can prevent others

130 Software may benefit from patent protection if it forms a part of a product or process constituting an invention, and such a software-related invention is not excluded from patentability under applicable law (see generally: Kenneth Nichols, Inventing software: the rise of "computer-related" patents (Quorum Books, Westport, 1998). More precisely, software often constitutes an element of an invention consisting of some "functional interrelation between technical components of a system, *e.g.*, the architecture of a processor and the particular way of processing data in such a processor." (Yannis Skulikaris, *Software-Related Inventions and Business-Related Inventions; A review of practice and case law in U.S. and Europe*, Patent World, February 2001, 26.) Notably, copyright protects expression (form of the work), whereas patent may be granted for the practical application of innovative ideas effecting in a product or process (Ian J. Lloyd, Information Technology Law, 308 (Butterworths, London, Edinburgh, Dublin, 2000, 3rd ed.)). See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 131.

131 Certainly, if a patent holder is at the same time a party to a Free Software license it is possible to argue that the patent holder has granted an implied or explicit patent license. We should note that only some of Free Software licenses explicitly cover other exclusive rights of the licensor than copyright, that may attach to a computer program (examples include the MPL, GPL, and Eclipse Public License), while others may contain only implicit patent licenses (see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004)). Still, many Free Software licenses are limited to copyrights only. Thus, even the licensor could attempt to enforce the exclusive rights that were not covered in the license to prevent licensees from exercising their freedoms (although this might constitute an abuse of rights, or be otherwise barred by the applicable law).

from using, developing, or distributing programs that are related to the patented product or process in a way covered by patent claims.

Obtaining patent protection requires application and registration at patent offices. The application, registration, and patent attorney fees may constitute an obstacle in applying for patent protection for some applicants. Additionally, patents are granted usually only if the reviewer of the application finds patentable subject-matter in it.[132] Patentability of software-related inventions depends on the applicable national law, and patents are granted for a specific jurisdiction.[133] This means that patents allow their holders to restrict user freedoms in some jurisdictions only.[134] It also means that patent holders may have difficulties in enforcing their patents against Free Software users scattered accross many jurisdictions. However, we are not aware of any major jurisdiction that would completely forbid granting of software-related patents in one way or another. Such patents have been widely recognised in the U.S.[135] In Europe, software-related patents have been granted

132 Some patent offices perform only a partial check, while some of them do not perform any examination at all, which means that the validity of the patents is checked only in case the patent is enforced and the alleged infringer defends in a court. See: WIPO (Standing Committee on the Law of Patents), *Report on the International Patent System*, (3 February 2009, SCP/12/3 Rev. 2), p. 53.

133 Some vague legal guidelines for software patentability may already be found on an international level, within the framework of WTO in TRIPS Art. 27 providing for a patentability of inventions in all fields of technology. Partially in the light of TRIPS Arts. 27(2) and 27(3) which do not mention software as capable of being excluded from patentability (Daniele Schiuma, *TRIPS and Exclusion of Software "as Such" from Patentability*, Vol. 31, No.1, IIC International Review of Industrial Property and Copyright Law 36, 40 (2000)). On the other hand; as Tripathi and others rightly point out, patenting things that do not meet general patentability requirements and are not inventions is not the objective of TRIPS, although Art. 1 allows to establish a more extensive protection than the one provided for by the agreement (R C Tripathi *et al.*, *Patenting of Computer Software: Status and Approach*, Vol. 7 Journal of Intellectual Property Rights 128, 129 (2002), adding that in the early 90s, when TRIPS was drafted and discussed software was mostly considered not to constitute inventions but algorithms, at best just discovered (*Id.* at 130)).

134 Generally in order to obtain a patent effective worldwide, the inventor would have to apply in each national jurisdiction separately. However, there are means to obtain patent for multiple jurisdictions as a result of single or simplified application (which are a result of international treaties such as the PCT or EPC).

135 In the U.S. for a long time, the ruling decision in the American debate on software patentability has been *State Street Bank & Trust v. Signature Financial Services*, 149 F.3d 1368 (Fed. Cir. 1998), cert. denied, 119 S.Ct. 851 (U.S. Jan 11, 1999). Court of Appeals for the Federal Circuit was presented with a computerized algorithm for managing an investment fund structure and held that it constitutes a patentable subject matter, which should be evaluated under the usual test of usefulness, novelty and non-obviousness (Peter Toren, *Software and Business Methods are Patentable in the U.S. (Get over it)*, Patent World, September 2000, 8). See also: Christopher L. Ogden, *Patentability of Algorithms After State Street Bank: The Death of the Physicality Requirement*, No. 10 Vol. 82 Journal of Patent and Trademark Office Society 721, 724 *et seq* (2000). In a recent case, however, *In re Bilski*, the Federal Circuit might have started a more rigid approach towards software patentability (545 F.3d 943 (Fed. Cir. 2008) (en banc)). See, *e.g.*, Patently O, http://www.patentlyo.com/patent/2008/10/in-re-bilski.html).

under the EPC.[136] Although the so-called Software Patents Directive proposal was rejected, some national patent laws in Europe have been applied in a way to allow patentability of software-related inventions.[137] Additionally, no restrictions on patents provided in patent laws known to us could allow Free Software users to disregard the threat of patents.[138]

It follows that we have to include an additional rule in the framework, the "software-related patents" rule.

> **Software-related patents**: Users are *not allowed* to exercise some or all of the six activities covered in the FSD to the extent that and in a jurisdiction where such activities constitute an infringement of a patent, unless they obtain consent of a patent holder.
> In particular, it is not allowed to use, develop, or distribute Free Software in jurisdictions where the use, development, or distribution (respectively) of that Free Software constitutes a patent infringement.

Software patents cover all four combinations presented in Table 3.1. They allow patent holders to restrict user freedoms regardless of whom a user obtained Free Software, and regardless of what kind of Free Software was obtained (original or improved). Certainly, software-related patents cover these combinations only to the extent they apply to Free Software.[139] So, each

136 The European Patent Convention of 1973 (EPC) excludes computer programs from the understanding of inventions in Art. 52(2)(c). However, pursuant to Art. 52(3), patentability is excluded only to the extent to which an application relates to computer program "as such". However, relying on the case law of Appellate Body, EPO has for a long time not rejected software-related applications straight away. The exception of EPC Art. 52(3) has been generally understood by EPO as excluding non-technical software-related inventions from the understanding of "invention". To pass the test for patentability, "technical effect" has to go beyond mere interaction between hardware and software (Skulikaris, FN 130 at 28). In the late 90s this term was broadened in the Appellate Body decision T 1173/97 to include "a computer program claimed by itself" if only the technical effect was present (*International Business Machines, Corp./Computer program product*, Decision of Technical Board of Appeal 3.5.1 dated 1 July 1998, T 1173/97 (OJ 10/1999, 609)). See: EPO, *Guidelines for Examination in the European Patent Office (2009)*, at: http://www.epo.org/patents/law/legal-texts/guidelines.html. See also: Daniel J. M. Attridge, *Challenging Claims! Patenting Computer Programs in Europe and the USA*, 1 Intellectual Property Quarterly 22, 44 (2001). But see: EPO, *Pending referral to the Enlarged Board of Appeal (G 3/08)*, at: http://www.epo.org/topics/issues/computer-implemented-inventions/referral.html.

137 Although Polish Industrial Property Law does not allow to grant patents for computer programs there have been many patents granted in one way or another related to products or processes that use computer programs (see, *e.g.* PL 123 820, published on 25 September 1984, PL 116 724, published on 31 March 1983).

138 Some Free Software licenses contain wording aimed at minimizing the threat of patents. Their intended effect is to discourage patent litigation (See: Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 42. For other types of defences against patents on the Free Software scene see *id*. 89, 95-96).

139 According to some studies, the Linux kernel infringes a few dozen of patents. See, *e.g.*, eWEEK, *Open-Source Insurance Provider Finds Patent Risks in Linux*, http://www.eweek.com/c/a/Linux-and-Open-Source/OpenSource-Insurance-Provider-Finds-Patent-Risks-in-Linux/.

Free Software program has to be scrutinized separately under each applicable law and patents granted in the jurisdiction in question. With these reservations, software-related patents restrict the freedoms that result out of *the grant of freedoms* by making it insufficient for a user to obtain a consent (license) of the copyright holder for the exercise of the freedoms. The consent of the patent holder is made an additional necessary condition. It means that the rule of software-related patents is in a direct relation with *the grant of freedoms*.

We include in the model of the framework software-related patents (RU9) and the relation between this rule and *the grant of freedoms* (RE14).

(5) Contracts with distributors

In Subsection 2.1.2 we indicated that some distributors of Free Software use restrictive distribution and service contracts. We should remind that, for example, users of some commercial GNU/Linux distributions are required to undertake contractually towards distributors (1) not to use certain software or (2) to use the software only in a particular way. Some users also undertake to refrain from modification or distribution of Free Software. Many of such obligations are not subject to any extreme sanctions. Usually, a user risks only losing warranty or services related to the software. However, this can be sufficient to prevent many users from exercising their freedoms. Certainly, not all distributors affect users only negatively. Some of them, in exchange for user freedoms, provide additional warranty for software, thus minimizing the risk resulting for users from *the hacker immunity*. We notice that such contracts do not affect the status of Free Software from the point of view of the FSD; Free Software programs subject to such contracts remain Free Software. However, by entering into such contracts, users invite new rules in the framework that affect their freedoms regulated by the rules identified so far.

Given the autonomy of parties to a contract that follows from contract law, there are practically no limits as to what the new rules could be. Here, we provide just four examples. First, users may contractually undertake not to exercise some or all of the six activities that constitute their freedoms. Second, they may in particular undertake not to fork the program. Third, it is possible to limit contractually the rules of *copyleft*. Namely, the distributor delivers improvements under a Free Software license to a user (thus performing *copyleft* obligations), but the user contractually undertakes not to distribute them, or not to use them in some other way.[140] Fourth, distributors could either provide additional warranty, or explicitly extend *the hacker immunity* that follows from the license of the distributed program on themselves (in such cases user freedoms are not restricted, *ceteris paribus*). Notably, any such new contract rule affects only users that enter into the contract.

140 According to the FSF, if the program is under the GPL, "[the licensee is] not allowed to distribute the work on any more restrictive basis" (Free Software Foundation, *GNU General Public License Frequently Asked Questions*, http://www.gnu.org/licenses/gpl-faq.html#DoesTheGPLAllowNDA). Such a contract would be thus a breach of the GPL. However, it would still bind the user *vis-à-vis* the licensee (the distributor).

Other users are not affected by the contract, and they may continue to exercise their freedoms (*ceteris paribus*).

Certainly, even though the autonomy of the parties is the basic principle of the contract law, there are many laws that allow to interfere with a concluded contract. Consumer protection and competition laws are the most prominent examples. Users are definitely able to invoke such laws against distributors who attempt to impose restrictive contracts on them. But we find it unlikely that they would allow to impose on distributors an obligation to enable users to exercise their freedoms as a remedy. So, we have to conclude that the freedoms can be restricted contractually despite such laws and in the model of the framework we include a general rule that follows from contracts with distributors.

> **Contracts with distributors**: A user is *allowed* to exercise all six activities covered by the FSD as long as the user does not contractually undertake not to exercise some or all of them.
> In particular, a user can undertake to use the software in a specific way only, or not to become a developer or a distributor of a Free Software program.

The above rule affects only these two combinations in Table 3.1 that regulate Free Software (original or improved) obtained from third parties – the distributors (the right column). Depending on the clauses of a particular contract with a distributor, this rule affects separately *the grant of freedoms*, *the right to fork*, *copyleft*, *the hacker immunity*, or all these rules at the same time. The extent that these rules are affected also depends on clauses of a particular contract.

We include in the model of the framework:

(1) contracts with distributors (RU10),
(2) the potential relation between contracts with distributors and *the grant of freedoms* (RE15),
(3) the potential relation between contracts with distributors and *the right to fork* (RE16),
(4) the potential relation between contracts with distributors and *copyleft* (RE17), and
(5) the potential relation between contracts with distributors and *the hacker immunity* (RE18).

(6) Liability rules

Liability rules provide for remedies in case of non-merchantability or unfitness of products for agreed or communicated purposes, other breaches of contract, torts, or infringement of third party rights. They also regulate whether and to what extent a person who otherwise would be liable is allowed to waive or escape such liability. We notice that the liability rules materially affect *the hacker immunity* (and it is the only rule in the framework affected by liability rules). Namely, liability rules may limit the scope of *the*

hacker immunity and bring back an important demotivator, the liability, into the path of Free Software actors such as developers and distributors. In particular, liability rules affect the readiness of developers to contribute new pieces of Free Software and the readiness of distributors to offer Free Software to users.

Liability rules are provided for by the applicable national law. Each national law strikes a balance between interests of all actors at a different point. However, we do not expect that any national law could completely prohibit warranty disclaimers or liability limitations.[141] Usually, liability rules do not allow to waive liability in some circumstances only, such as if the damages were caused due to gross negligence or wilful behaviour, as well as in relations with consumers. At the same time liability rules contain many exceptions that can be used to avoid liability or transfer it to another entity. For example, developers could hide behind the "corporate veil" of a legal person, such as a company.[142] Then, if the company distributed software that caused a damage not covered by *the hacker immunity*, the company would be liable for the damage *vis-à-vis* the users.

Much of the above dilemma is resolved in the Free Software scene using contracts. Many distributors offer additional contractual warranties or services related to Free Software in exchange for remuneration. By exercising such warranties or purchasing such services, users are able to have many defects in Free Software promptly removed. A prompt removal of defects means that the users are less likely to seek damages against the actors in the Free Software scene or to seek other legal recourse. The fact that under *the grant of freedoms* everyone is allowed to offer a warranty or a service in relation to a Free Software program stimulates the operation of the market of these warranties and services. However, if this market was prevented from operation, users would be more likely to litigate against the actors. Liability would become a real threat, which would demotivate the actors from developing and distributing Free Software despite *the hacker immunity*.

Nevertheless, liability is always a potential threat, irrespective of the operation of the market described above. The mere fact that a third party (a distributor) grants a warranty or stipulates to provide services related to Free Software does not affect general liability rules between users and those who actually cause damages. Generally, such warranties or services provide

141 See, *e.g.*, Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7 (arguing that the effectiveness of warranty disclaimers and liability limitations in Free Software licenses depends on whether they create a contractual relationship, not a condition on the use of software (at 30, 33); citing UCC and UCITA requirements for effectively waiving express and implied warranties). See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 78 *et seq*.

142 See: Wikipedia, *Corporate veil*, http://en.wikipedia.org/wiki/Corporate_veil.

only for an additional party a user could turn to in case of defects. The general liability rules that allow to turn to the person who actually caused the damage remain largely unaffected. This means that the hacker immunity is subject to liability rules, that is by all rules that regulate liability in the applicable law. So, we include in the framework liability rules that limit *the hacker immunity*.

> **Liability rules**: Users are *not allowed* to claim liability related to Free Software as long as the applicable law does not provide to the contrary, or it provides for exceptions that allow to avoid liability.

Precisely speaking, there are usually many rules that regulate liability. For example, these are: rules of statutory warranties, contractual liability rules, rules of torts, *etc.* In this thesis we group all of them in one class called "liability rules" and we include this class in the framework, as if it was a single rule. This is possible, because every liability rule is generally in the same relation with the hacker immunity. Namely, such a rule regulates (and most possibly restricts) *the hacker immunity* in all four combinations presented in Table 3.1.

We include in the model of the framework liability rules (RU11) and the relation between this class of rules and *the hacker immunity* (RE19).

(7) Non-legal regulators of software

So far we focussed on the identification of legal rules. However, the freedoms could also be subject to limitations or restrictions that follow from non-legal regulators, *i.e.*, (1) the architecture, (2) the norms, and (3) the market. Usually, it is not easy to indicate one of these regulators as an isolated source of a particular restriction or limitation. It should be definitely expected that many restrictions and limitations are a result of various regulators acting together. Here, it is important to account for the resulting non-legal restrictions and limitations in the form of rules that are included in the model of the framework. Below, we provide four examples of such restrictions and limitations. Then, we identify rules that follow from them.

First, some vendors make Free Software programs available as a part of a service[143] or of an embedded device.[144] In such a case, it is possible for them to control the exercise of the freedoms using the architecture of the service or of the device. Namely, they can employ technical means that make it

143 There is a growing trend of delivering software not as a product, but as a service. See: Wikipedia, *Cloud computing*, http://en.wikipedia.org/wiki/Cloud_computing. See also: autonomo.us, *Franklin Street Statement on Freedom and Network Services*, http://autonomo.us/2008/07/franklin-street-statement/.

144 There is a growing trend of embodying Free Software in various devices. See, *e.g.*, Wikipedia, *Tivoization,* http://en.wikipedia.org/wiki/Tivoization, see also: Wikipedia, *Android (mobile device platform)*, http://en.wikipedia.org/wiki/Android_(mobile_device_platform).

impossible or impracticable for users to use modified or third-party Free Software programs together with the respective service or device. As a result, users remain legally allowed to exercise the freedoms, but they have difficulties in using the software for purposes other than contemplated by the distributor. More importantly they are not able to study, to adapt, and to improve the software effectively. Obviously, they are unable to redistribute, and release the software as well (since it makes not much practical sense). Certainly, users could exercise their freedoms in an unencumbered way only if they were able to reconstruct the whole infrastructure used to provide the service, or if they were able to manufacture a substitute embedded device. This makes the exercise of the freedoms unlikely.

Second, the practice shows that average users do not exercise the freedoms, at least not to the full possible extent. Instead of, *e.g.*, improving programs on their own, many users rely on certain developers and distributors of Free Software (such as renowned firms or communities that develop and distribute GNU/Linux distributions). Such users do not modify the software if they identify bugs, require some lacking functionalities, or require a higher degree of interoperability. Rather, they send bug reports or feature requests to these developers and distributors. These users do not develop the program on their own if they see that the development is not organized according to their will. This means that many users turn into passive audience, such as the users of proprietary software, despite they are legally allowed to be active actors. Obviously, the fact that the users have the freedoms from the legal point of view, does not mean that they will exercise the freedoms at all times.

Third, it has been observed in practice that *copyleft* does not necessarily enable users to exercise their freedoms in the improvements of Free Software made available as a result of this rule. Harald Welte claims that he managed to enforce *copyleft* in over 100 cases.[145] However, he noted that software that was made available as a result of this enforcement was mostly useless. Welte indicates the following reasons: (1) low quality of the source code (poor coding style, API problems, lack of portability); (2) outdated source code (written against old versions of the original program); (3) *copyleft*-avoidance practices such as "binary kernel modules". This shows that the law alone (as expressed in a *copyleft* clause) is not a sufficient regulator that could direct anyone to develop good-quality code, to write it against recent versions of programs, or not to attempt to game the boundaries of the scope of *copyleft*. We anticipate that such a regulation could be a result of norms or of the market. However, Welte's observations already allow us to conclude that there are no such rules that regulate *copyleft* infringers effectively.

Fourth, assume a person makes an improvement, such as a patch or a feature upgrade, to a Free Software program. Assume also that the improvement is of good quality, works with the current version of the program, and

145 Harald Welte, *Some more thoughts on the results of GPL enforcement*, at: http://gnumonks.org/~laforge/weblog/2006/10/30/#20061030-gpl-devices.

that it is provided with the whole source code. Then, the improvement has to be properly integrated in the whole program. Only if it is properly integrated, a working new version of the program is created, and the freedoms to the new version may be exercised by users in practice. Otherwise, the program would not work at all, or it would work after each user invested a considerable technical effort. The integration is possible if only the development of software is organized properly. The organization is necessary because the integration requires proper skills of developers and cooperation between them. It means that there are transaction costs involved in the integration.[146] These transaction costs might prevent individual users from integrating the code on their own.

Here, we may conclude that non-legal regulators limit or may be used to restrict user freedoms. In the first example given above, the restriction is a result of the architecture as constructed by vendors of services or devices offered together with Free Software. The architecture has an impact similar to contracts with distributors, already described in one of the above paragraphs. So, we include in the framework, called "architectural restrictions".

> **Architectural restrictions**: A user *can* exercise all six activities covered by the FSD as long as the user is not prevented from it by technical means.
> In particular, technical means could prevent users from using, developing, or distributing a Free Software program.

Architectural restrictions cover all combinations presented in Table 3.1, since they can be applied regardless of whom distributes Free Software and what kind of Free Software is distributed. The restrictions are in a direct relation with *the grant of freedoms*. Namely, they are used to restrict the exercise of the freedoms granted by copyright holders to Free Software. We include in the model of the framework the architectural restrictions (RU12) and the relation between this rule and *the grant of freedoms* (RE20).

In the second example, we observe a limitation. Users exercise the freedoms in some situations only. We assume that the exercise of the freedoms comes at a price; it causes transaction costs. Namely, it requires technical knowledge necessary to create a working computer program out of an original Free Software program and improvements of the program. Also, the resulting software requires support and maintenance services. In such a case, users who exercised freedoms in the program on their own, would have to support and maintain it by themselves as well. Thus, in the second example

146 *Cf.* Ronald Coase, *The Nature of the Firm*, Economica, vol. 4, no. 16, November 1937 at 386. Transaction costs, generally, are costs that individuals bear while acting on a free market. According to Yochai Benlker a particular type of transaction costs present in the development of Free Software are "integration costs", *i.e.*, costs of controlling quality of contributions and integrating them in the whole product (Yochai Benkler, *Coase's Penguin, or, Linux and The Nature of the Firm*, 112 Yale Law Journal 369 (2002)). See also: Wikipedia, *Transaction costs*, http://en.wikipedia.org/wiki/Transaction_costs.

the limitation on the protection of user freedoms is a result of the market, which prevents rational users (*homo economicus*) form undertaking activities, of which costs outgrow benefits. This limitation affects *the grant of freedoms* and can be expressed in the following way.

> **Limitation of *the grant of freedoms* by the market**: A rational user will exercise the freedoms provided that the benefits for the user are higher than costs.

The limitation covers all combinations presented in Table 3.1 that are at the same time covered by *the grant of freedoms*. We note that this limitation does not always affect users who do not have to follow the cost/benefit calculus, such as the government or non-profit organizations. The limitation is in a direct relation with *the grant of freedoms*. We include in the model of the framework the limitation of the grant of freedoms by the market (RU13) and the relation between this rule and *the grant of freedoms* (RE21).

In the third example we also observe a limitation. It is due to insufficient market regulation or insufficient norms that could stimulate all users not to infringe *copyleft*. This leads to a situation wherein *copyleft* has to be enforced using the law. However, the law alone cannot direct *copyleft* infringers to write good quality source code and provide it in a way that allows to use the code by other users. This limitation directly affects *copyleft*.

> **Limitation of *copyleft* by the market and norms**: A licensee will perform the *copyleft* obligations provided that (1) the benefits for the licensee are higher than costs and (2) the non-performance of the obligations is contrary to norms that bind the licensee.

The limitation covers all combinations presented in Table 3.1 that are at the same time covered by *copyleft*. Consequently, the limitation is in a relation with *copyleft* only. It consists of two regulators: (1) the market, and (2) the norms. We note that the market limitation applies only to users who have to follow the cost/benefit calculus, while the remaining users are only subject to the norm that stigmatises *copyleft* infringement. We include in the model of the framework the limitation of *copyleft* by the market and norms (RU14) and the indirect relation between this rule and *copyleft* (RE22).

In the fourth example, there is another limitation. It is a result of transaction costs related to the integration of source code that comes from various sources into a finished working software product. If a user cannot bear these transaction costs, the user has to forgo the exercise of the freedoms in a particular improvement of Free Software. There is a general rule that follows from this example. It regulates all six activities covered by the FSD, not only improving Free Software. We call this rule "market limitation of freedoms".

> **Market limitation of freedoms**: A user will exercise the six activities covered by the FSD provided that the benefits for the user are higher than costs.
> In particular, a user will not become a developer or a distributor of Free Software if the development or distribution (respectively) is not viable economically.

Market limitation covers all combinations presented in Table 3.1, since it can be applied regardless of whom distributes Free Software and what kind of Free Software is distributed. The limitation is in a direct relation with *the grant of freedoms*. Namely, it limits the exercise of the freedoms granted by copyright holders to Free Software. Again, we note that the limitation applies only to users who have to follow the cost/benefit calculus. So, if a given user is able to undertake activities, of which costs outgrow their monetary benefits, and the exercise of a particular freedom leads to such an effect in the given circumstances, then the user will not be prevented from the exercise of the freedom by this limitation. We include in the model of the framework the market limitation (RU15) and the relation between this rule and *the grant of freedoms* (RE23).

In summary, as a result of the analysis of non-legal regulators, we include in the framework four rules and four relations. For the sake of clarity, we group all this rules in one class, to which we will refer to as "non-legal regulators" (RU12-15). This class of rules is in relation with *the grant of freedoms* (RE20, RE21, and RE23) and it is in relation with *copyleft* (RE22).

3.1.3 Conclusion on rules for software and relations between them

In Subsections 3.1.1 and 3.1.2 we identified 15 rules that regulate access to software. We also identified 23 relations between these rules. We provide a simplified graphical representation of the rules identified so far and relations between them in the Figure 3.5.

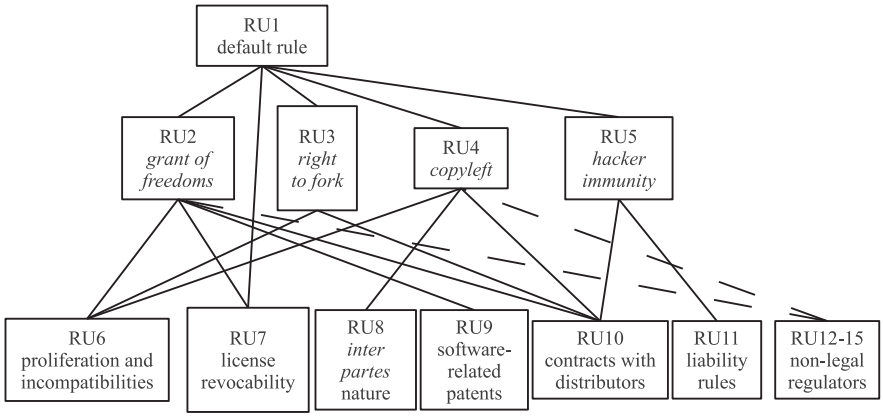

Figure 3.5: Rules for software and relations between them

For the sake of clarity, we performed the following simplifications in Figure 3.5. First, we group together all non-legal regulators in one item (as explained above). Second, we do not present all 23 relations, but only the relations we find most important for the purpose of our study. Namely, we omit relations between *the grant of freedoms*, *the right to fork*, and *copyleft*, as these relations

do not limit or restrict user freedoms. For all other identified relations, the reader should revert to the respective paragraphs above. The presented relations between rules are indicated using lines. Solid lines represent legal relations. Dashed lines represent relations between legal rules and the rules that follow from non-legal regulators.

In Figure 3.5 we present the rules grouped in three levels. On the first level there is *the default rule*. This is the rule which empowers copyright holders to grant or refuse user freedoms. On the second level there are 4 rules that grant or protect user freedoms (*the grant of freedoms*, *the right to fork*, *copyleft*, and *the hacker immunity*). On the third level there are 10 rules that constitute limitations or restrictions of user freedoms (since 4 non-legal regulators are grouped in one class, there are only 7 boxes presented on the third level). In this thesis we will focus on the third level, that is on the limitations and restrictions of the freedoms. For example, we will analyse how these limitations and restrictions are affected by software communities (in Chapter 4) and eGovernments (in Chapter 5). Also, we will propose improvements of the framework designed to address the limitations and restrictions (in Chapter 6).

Here, we may conclude that we identified rules that are related to the software scene and relations between them. We may also conclude that we include in the framework all the identified software rules and relations.

3.2 *Identification of rules for standards and relations between them*

It follows from our findings in Chapter 1 that there are two necessary conditions of user freedoms: (1) access to software, and (2) access to standards. All rules identified so far regulate access to software. So, we still have to identify additional rules that regulate access to standards. Then, we have to identify relations between them. Afterwards, we have to include these rules and relations in the model of the framework. We do all this in this section. First, we identify rules and relations that limit or restrict access to standards and lead to closed standards (3.2.1). Then, we identify rules and relations that regulate activities directed at removing the restrictions on standards and at making open standards (3.2.2). Finally, we present conclusions on the rules for standards and on the relations between them (3.2.3).

3.2.1 Rules that lead to closed standards

To put it in a nutshell, closed standards are protocols, interfaces, and data formats, of which the specifications are not generally available and that are subject to exclusive control.[147] Access to, and the use of, interoperability

147 In this thesis we treat any standard that does not meet the definition of an open standard, as a closed standard. The definition of an open standard was presented in Chapter 2, Section 2.3. This definition was taken from the EIF v. 1.0. Here, "available" means not only "accessible without limitations", but also "capable of being legally used without restrictions".

information of such standards is restricted. Restrictions that lead to closed standards are the result of three regulators: (1) the law, (2) the architecture, and (3) the market.[148] First, the law protects, in particular, trade secrets and patents. If interoperability information satisfies certain criteria, it becomes *a trade secret*. Alternatively, the use of the information may be protected by *a patent material to the standard*. So, the law can be used to restrict access to standards indirectly, by invoking trade secret or patent protection. Second, it is possible to design a program that uses a standard in a way to restrict access to the interoperability information. We refer to this as the *scrambling of interoperability information*. Scrambling makes it technically impossible or impractical to access the necessary information. Third, it is possible to make use of switching costs in order to *lock* users *in* to a particular closed standard. In such a way, standards are restricted as a result of the market regulation.

Generally, closed standards and Free Software exclude each other. Certainly, developers of Free Software programs sometimes attempt to design their programs to use closed standards. However, as a rule, the development, distribution, or use of Free Software that uses closed standards is either legally prohibited, technically impractical, or economically infeasible. This leads to two possibilities: (1) copyright holders of would-be Free Software programs that use closed standards are stimulated not to grant users their freedoms in such software, or (2) users are stimulated not to exercise the freedoms. Sometimes, the stimulation results in a situation that such software is not developed at all. Alternatively, it could be possible to grant users freedoms to a program that uses a closed standard despite the stimulation, but the regulators mentioned above would still prevent users from exercising their freedoms in whole or in part. In particular, would-be developers of such programs could not engage in the development of the programs, or would-be distributors could not distribute them. More precisely, the designer of the standard can either prevent them from doing so completely, or impose conditions incompatible with the freedoms. So, the general rule for closed standards that we include in the framework is "closed standards" (RU16).

> **Closed standards**: users of a program are *not allowed* to exercise (or *cannot* exercise) some or all of the six activities covered by the FSD, as long as the program uses a closed standard.
> In particular, developers are *not allowed* to (*or cannot*) freely develop Free Software that uses such a standard, and distributors are *not allowed* to (*or cannot*) freely distribute this software.

This general rule is in direct relation with *the grant of freedoms* (RE24). It is a complex relation that can be explained in the following way. Even though

148 As far as the fourth regulator, the norms, is concerned, we proceed under an assumption that it does not have a material impact on the regulation of standards. However, there are certainly some norms that regulate standards, in particular the norms followed by participants of standard setting organizations.

users can invoke a Free Software license against the licensors who attempt to restrict user freedoms, the users are still restricted in the exercise of the freedoms as long as the program in question uses a closed standard. Alternatively, because the standard is closed, no Free Software is able to use this standard without restrictions. So, closed standards either restrict *the grant of freedoms* or stimulate copyright holders not to grant the freedoms at all.

The rule of closed standards affects all four combinations presented in Table 3.1. Namely, it allows to restrict the exercise of the freedoms regardless of the fact from whom the program is obtained and regardless of whether the program is original or improved Free Software. However, this is the case only to the extent that the program uses (or rather attempts to use) closed standards. More precisely, this is the case when users attempt to both (1) exercise the freedoms, and (2) provide for interoperability using closed standards. In other cases, the rule does not apply.

We include in the model of the framework the closed standards (RU16) and the relation between this rule and the grant of freedoms (RE24). Below, we refine the rule of closed standards by elaborating on four issues that are the result of the three regulators described above. There are two issues related to the law: (1) trade secrets, and (2) patents material to standards. There is one issue that is related to the architecture: (3) scrambling of interoperability information, and there is one issue that is related to the market: (4) locking in.

(1) Trade secrets

Interoperability information related to a closed standard can constitute a legally protected trade secret of the designer of the standard.[149] Generally, the legal protection of the information as a trade secret is conditioned upon making it secret and on undertaking efforts to protect the secret.[150] Certainly, in order to restrict access to a standard with the use of trade secrets, the designer of the standard has to keep its specification private. However, the designer can make the specification available to others without renouncing trade secret protection; this is done using non-disclosure agreements ("NDAs"). An NDA obliges the recipient to keep the information confidential. Notwithstanding the making available under an NDA, the standard

149 "Trade secrets" is an American legal term, but it has been recognized throughout the world. In various continental systems the terms "confidential information" or "know-how" are also used. TRIPS regulates this matter in Sec. 7, Art. 39 titled "Protection of Undisclosed Information". In national laws that follow the civil law tradition, trade secrets are usually protected under general provisions of civil codes, or under the laws on unfair competition.

150 Carey R. Ramos, David S. Berlin, *Three Ways to Protect Computer Software*, 16 No. 1 Computer Lawyer 16 (1999); Victoria A. Cundiff, *Protecting Computer Software as a Trade Secret*, in: 507 Practising Law Institute, 18th Annual Institute on Computer Law 761 (1998); Alois Valerian Gross, *What is Computer "Trade Secret" under State Law*, 53 American Law Reports 4th 1046. See also: Van Lindberg, Intellectual Property and Open Source, O'Reilly 2008, 103 *et. seq*.

remains a closed standard, because NDAs oblige parties to protect the information from becoming generally available and used. They may also impose other restrictions on the use of the information by the recipient.

Trade secret protection terminates (1) if the designer fails to protect it adequately or reveals the protected information, (2) if the information becomes generally available without a breach of an NDA, or (3) if it becomes available in another lawful way.[151] Afterwards, the information can be legally used to provide for interoperability, specifically for interoperability of a Free Software program with other programs that use the given standard. Conversely, as long as a standard is protected by trade secrets, it is not possible to implement the standard in a Free Software program. Obviously, the designer of the standard could allow certain persons to exercise some degree of freedom, but practically there is no possibility to allow a person to exercise all four freedoms as defined by Stallman while maintaining trade secret protection for a standard used in a Free Software program at the same time.[152] In particular, development or distribution of a Free Software program that uses a standard protected as a trade secret without a breach of trade secrets law is generally illegal, or at least impracticable. So, the general rule of closed standards may be refined in the following way.

> **Trade secrets**: As long as compliance with a standard requires the use of information that constitutes a trade secret, users of programs that use the standard are *not allowed* to exercise such activities covered by the FSD that constitute a breach of the trade secret.
> In particular, developers are *not allowed* to develop Free Software that uses such a standard, and distributors are *not allowed* to distribute this software.

This rule (RU16a), as a refinement of the general rule of closed standards, is in relation with *the grant of freedoms* (RE24) in the way already described above. The coverage of this rule of the combinations presented in Table 3.1 as compared to the general rule of closed standards is further limited to cases when the standard is protected by trade secrets. If the trade secret protection does not apply, user freedoms are not affected by this rule.

(2) Patents material to standards

Patents can be used to restrict access to standards much more effectively than trade secrets. We already outlined the basic rules of patent protection when discussing the rules for software. There, we focussed on software-

151 Applicable laws can differently regulate this issue. We may assume that in most major jurisdictions the use of a trade secret obtained in an unfair way (*e.g.*, "industrial espionage") would rather not be legal, while the reconstruction of the information using fair means (*e.g.*, reverse engineering) would not be prohibited generally.

152 David Bender, *Trade Secret Implications of Open Source Licenses*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 129; Stephen J. Davidson, Gabriel Holloway, *Protecting Trade Secrets in an Open Source Environment*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 143.

related patents, of which the claims allow to restrict user freedoms because they extend to (a) products including a Free Software program or (b) processes performed by such a program. Here, we focus on patents material to standards, which allow to restrict the freedoms because their claims extent to products or processes designed in compliance with a standard. If a Free Software program is included in such a product or performs such a process, its development, distribution, or use can constitute patent infringement (depending on the patent in question and the applicable law).

From this point of view it is necessary to add that patents grant the holder the monopoly to use the patented invention (product or process), as well as many substitute products or processes.[153] In case the subject-matter of a patent is an interface, protocol, or a data format, then (depending on the exact scope of patent claims) the patent holder may restrict the use of Free Software programs that interoperate using such a patented subject-matter, even if the programs have been independently created. Obviously, the patent holder can also restrict the development and distribution of such programs.[154] The fact that patents material to standards can be obtained creates an incentive for existing or prospective patent holders to direct standardization towards adopting closed standards covered with their patents. As a result many parties could not avoid using the standard, while at the same time they would have to obtain the patent holder's authorization to do so.[155] From the point of view of user freedoms this means that the exercise of the freedoms would be subject to such an authorization, as long as the exercise involved the use of the closed standard covered by a patent. So, the general rule of closed standards may be refined in the following way.

> **Patents material to standards**: As long as compliance with a standard requires the use of a subject-matter covered by a patent, users of programs that use the standard are *not allowed* to exercise such activities covered by the FSD that constitute a breach of the patent.
> In particular, developers are *not allowed* to develop Free Software that uses such a standard, and distributors are *not allowed* to distribute this software.

153 This is a result of the so-called "doctrine of equivalents" (See, *e.g.*, Wikipedia, *Doctrine of equivalents*, http://en.wikipedia.org/wiki/Doctrine_of_Equivalents. See also: Timothy R. Holbrook, *The paradoxical nature of U.S. patent scope*, in: Maciej Barczewski et al. (eds.), When Worlds Collide: Intellectual Property, High Technology and the Law (Wolters Kluwer 2008), 65.

154 The restriction is possible even if the specification of a patented interface, protocol, or data format is publicly available. Conversely, patent claims are usually not expressed in a way that constitutes a complete specification of protocols, interfaces, or data formats covered in whole or in part by them. Thus, patents may form an additional layer of protection on top of the protection resulting from the mere fact that the interoperability information is kept secret. The practice of not disclosing the specifications or source code implementations in patents is contrary to the basic *quid pro quo* rule of patent law, which conditions the patent grant on the disclosure of the invention to the public.

155 See WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.

This rule (RU16b), as a refinement of the general rule of closed standards, is in relation with *the grant of freedoms* in a way already described above (RE24). The coverage of this rule of the combinations presented in Table 3.1 as compared to the general rule of closed standards extends only to cases when the standard is protected by patents. If the patent protection does not apply, user freedoms are not affected by this rule. We remark that patent holders can authorize users to exercise the freedoms, so patents restrict the freedoms only if the patent holders refuse to provide such an authorization (patent license). But there is no obligation to license patents to users of Free Software. The patent holders may also place conditions on the exercise of the freedoms, or even prohibit the exercise of the freedoms completely (under particular circumstances).

(3) Scrambling of interoperability information

Interoperability information necessary to use a standard is initially known to the designer of the standard only. Sometimes, the designers of an interface, protocol, or data format do not even embody the information in any documentation (such as a specification) but merely implement the standard directly in a program that uses it. If, in such a case, the program is distributed in a binary form only, then the reconstruction of interoperability information is usually impossible or impractical. Often, even if source codes of the program are available, the information cannot be easily reconstructed from them. Actually, the information expressed in any other form than a complete and accessible specification is effectively *scrambled*, *i.e.*, it cannot be easily used to implement the standard in an independently developed program.

The advantage of scrambling over trade secrets or patents is that it only requires protecting the information physically and designing one's products in a specific way. It does not require in particular any specific conduct regulated by law. Certainly, scrambling alone may not be used to prevent attempts to extract the hidden interoperability information or to prevent the design of Free Software that uses the extracted information. Still, scrambling may be used to make it impossible, impractical, or at least burdensome to access the interoperability information. Usually, scrambling is used together with trade secrets or patents. In such a way, it provides for an additional layer of restrictions on an already closed standard. So, the general rule of closed standards may be refined in the following way.

> **Scrambling of interoperability information**: As long as the interoperability information for a given standard is not expressed in a complete and accessible standard specification, users of programs that use the standard *cannot* exercise some or all activities covered by the FSD with regard to these programs.
> In particular, developers *cannot* develop Free Software that uses such a standard, and distributors *cannot* distribute this software.

This rule (RU16c), as a refinement of the general rule of closed standards, is in relation with *the grant of freedoms* in a way already described above (RE24).

The coverage of this rule of the combinations presented in Table 3.1 as compared to the general rule of closed standards extends only to cases when the standard is not expressed in a complete and accessible specification (that is to cases when the access to interoperability information is restricted or limited using technical means). If this is not the case, user freedoms are not affected by this rule.

(4) Locking in

Access to interoperability information for a standard is a necessary condition for the development of a program that uses the standard. Consequently, it is a necessary condition for the distribution of such a program to users. However, it is not a sufficient condition that users will prefer programs that use open standards instead of programs that use closed standards. Sometimes, the costs of using an open-standards-based program are higher than the benefits. An important part of such costs are *switching costs*. Switching costs are a type of transaction costs related to the fact that users have to use programs that properly interoperate with other elements of their system, as well as with programs used by other users. Thus, a user will use a program that implements a given standard if (1) the rest of the user's system already supports the standard or may be easily adapted to support it, and (2) if other users already use programs that support that standard or may be easily adapted to support it.

Should the overall switching costs be high, users might be effectively stimulated not to switch to programs that use other standards than the already popular standard. If such a popular standard is an open standard, user freedoms are not restricted (*ceteris paribus*). However, if such a popular standard is a closed standard, switching to programs that use another standard is not viable economically. This means that users are locked in to programs that are designed to interoperate using that standard. As a result of a *lock-in* users cannot afford abandoning a particular closed standard. It follows, that if a designer of a closed standard succeeds in making its own closed standard popular, the designer can then effectively restrict user freedoms.[156] For a successful lock-in, the popularity has to be coupled with the

156 User choices of programs are to a significant extent driven by network effects. As defined by economics, network effects cause goods to gain value, as perceived by their users each time the number of the users increases (See, *e.g.*, Wikipedia, *Network effects*, at: http://en.wikipedia.org/wiki/Network_effects). Network effects exist, for example, in markets of products that allow people to communicate with themselves. *Ceteris paribus*, if it is possible to offer a product that makes it possible to connect to a bigger number of people, then such product will prevail over the competition. The network effects of such a product would be higher than that of competing products. Conversely, if two products have the same network effects (*i.e.* allow to connect to the same people) then their market success will be determined by other factors, such as the quality of communications they provide. See: Arun Sundararajan, *Network Effects*, at: http://oz.stern.nyu.edu/io/network.html. Network effects are strongly related to Metcalfe's law. As explained in the Wikipedia, Metcalfe's law states that the value of a telecommunications network is proportional to the square of the number of users of the system (Wikipedia, *Metcalfe's law*, at: http://en.wikipedia.org/wiki/Metcalfe%27s_law).

application of at least one of the already identified rules that lead to closed standards. Otherwise, developers could succeed in development of Free Software that is able to interoperate with the popular standard, and the freedoms would not be materially restricted. So, the general rule of closed standards may be refined in the following way.

> **Lock-in**: If the costs (in particular switching costs) of using a Free Software program that does not support a particular closed standard are higher than benefits, users *will not* exercise their freedoms in such a Free Software program.
> In particular, developers *will not* develop such Free Software, and distributors *will not* distribute it.

This rule (RU16d), as a refinement of the general rule of closed standards, is in relation with *the grant of freedoms* in a way already described above (RE24). The coverage of this rule of the combinations presented in Table 3.1 as compared to the general rule of closed standards extends only to cases when the standard is restricted using above-described market regulation. We note that although this rule is a result of the market, it does not affects only such users who have to follow the cost/benefit calculus. Namely, if such users do not switch to open standards due to the fact that their costs outgrow monetary benefits, other users who could afford the switch cannot do so if they want to interoperate with the former users.

Here, we may conclude that the general rule of closed standards (RU16) consists of four specific rules (RU16a-d), each a result of a different issue discussed in this subsection. Each of these specific rules is in the same relation (RE24) with the grant of freedoms as described at the beginning of this subsection.

3.2.2 Rules that lead to open standards

Access to standards may be restricted using (1) the law (*i.e.*, trade secrets or patent law), (2) the architecture (*i.e.*, scrambling of interoperability information), and (3) the market (*i.e.*, locking in). Usually, the access is restricted using a combination of all these three regulators. Obviously, from the point of view of user freedoms access to standards should not be restricted and the standards should be open. Thus, apart from (1) the rules that allow to restrict access to standards identified above, we have to identify (2) rules that allow to access standards and make them open. All these rules should be included in the framework. We may identify the latter rules by analysing the activities that are usually attempted in order to remove some or all restrictions on standards. These activities are: (1) reverse engineering, (2) standard setting, and (3) claiming essential facilities. Below, we elaborate on all three of them.

(1) Reverse engineering

In the previous subsection we indicated that it is possible to make a standard closed by scrambling interoperability information. Namely, the specification of the standard is not made available to users, only the program that uses the standard is available. In such a situation, a sufficiently skilled person may still be able to extract the necessary interoperability information by reverse engineering the program. To reverse engineer a program means to analyse it, and to extract certain information.[157] The extracted information can be in particular the interoperability information, which then can serve to reconstruct the specification of the standard embodied in the reverse-engineered program. After the reconstruction, the specification can be used in the development of another program that is able to interoperate using that standard.

Generally, there are two methods to reverse engineer a program: (1) analyse its source codes, and (2) analyse how the program works (its input and output data). Obviously, the first method can be used if only the source codes are available. If they are not readily available, it may be possible to reconstruct them during decompilation of binaries.[158] The second method does not require source codes or decompilation, it is performed by running the reverse-engineered program on different sets of data. However, any method of reverse engineering does not guarantee completeness of the interoperability information extracted. Even if source codes of a program are available, they may be not sufficient to prepare a complete and useful specification of a standard used in the program. Also, the developer of the reverse engineered program could release a new version of the program that uses a modified standard. Provided that all or most of the users of the old version switch to the new one, the information reverse-engineered from the older version of the program becomes obsolete.

Apart from technical constraints, reverse engineering is legally limited as well. Proprietary licenses usually attempt to prohibit reverse engineering.

---

157 In a more general context, "[r]everse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction." E.J. Chikofsky, J.H. Cross II, *Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software*, IEEE Computer Society: 13–17 (January 1990), as quoted by the Wikipedia (Wikipedia, *Reverse engineering*, at: http://en.wikipedia.org/wiki/Reverse_engineering).

158 Precisely speaking, apart from decompilation, a person may also perform disassembly. The result of decompilation is a source code, while the result of disassembly is an assembly code. Assembly code is a human-readable expression of a computer program in a low-level programming language, as opposed to source code, which is an expression in a high-level language. For the sake of simplicity, we do not distinguish between decompilation and dissasembly. We may here assume that there are no differences between source code and assembly code, which would be material from the point of view of the construction of the model of the current regulatory framework of Free Software. Both methods may be used to reconstruct a specification of the standard used by the disassembled (or decompiled) program. (See: Wikipedia, *Reverse engineering*, at: http://en.wikipedia.org/wiki/Reverse_engineering).

Towards_an_Improved_def.indd 104 23-02-2010 10:49:23

The enforceability of such restrictive provisions varies across jurisdictions.[159] Also, elements extracted from a program during reverse engineering may be still subject to copyright, trade secrets, or patent protection. Additionally, reverse engineering is usually subject to strict legal regulation, and if improperly performed it could give raise to civil or criminal liability.[160] This requires extreme diligence during the process, for example by using methods such as "clean room design".[161] As a result of such legal considerations, it takes much efforts to reverse engineer a program in such a way that a program developed according to the reconstructed specification could be legally used. In case of a patented standard, it is quite probable that this would still require authorization of the patent holder.

Here, we may conclude that reverse engineering is not a sufficient method of making a standard open from the point of view of user freedoms. It does not affect materially any of the rules identified above that lead to closed standards. In particular, it does not guarantee that the specification reconstructed using the extracted interoperability information is complete, useful, and that it can be used legally and without other restrictions. Additionally, the use of such Free Software programs may be restricted with third party patents, even if the reverse engineering is successful. Consequently, Free Software programs that are designed according to such reconstructed specifications may work improperly. Also, their use could lead to liability of users, developers, or distributors. Additionally, the exercise of the freedoms in such a program may still be subject to authorization of the holder of patents material to the standard.

Since reverse engineering does not materially affect any of the previously identified rules that lead to closed standards, we do not find any additional rule that follows from reverse engineering that could be included in the framework.

---

159 See, *e.g.*, Pamela Samuelson, *Reverse Engineering Under Siege*, 10 Communications of the ACM 15 (2002).

160 For example, according to Polish law under certain circumstances it is possible to decompile a program in order to provide means for interoperability (Polish Copyright Act Art. 75, which implements Software Directive Art. 6). But the information obtained during reverse engineering may constitute a trade secret. Under the Polish Unfair Competition Act Art. 11.1 it is an unfair competition tort to communicate, reveal or use trade secrets of another entrepreneur, or to acquire them from an unauthorized party if it endangers or threatens the interests of the entrepreneur. It is not clear how this provision relates to the Copyright Act. Additionally, the use of information extracted from a protected device may constitute a criminal offense. According to art. 267.1 of the Polish Criminal Code it is a crime to obtain without authorization an information not addressed to the person, by opening a closed letter, connecting to a wire transmitting the information or by circumventing electronic, magnetic or other particular security mechanism. It is not clear how this provision relates to decompilation right as well.

161 See, *e.g.*, Wikipedia, *Clean room design*, at: http://en.wikipedia.org/wiki/Clean_room_design.

(2) Standard setting

Naturally, the source of the most complete interoperability information is the designer of the protocol, interface, or data format in question. Under certain circumstances, the designer might be interested in making the specification of the standard available. Indeed, there are numerous examples of designers cooperating by exchanging technical specifications. They usually do it within standard-setting organizations.[162] There, they attempt to reach consensus over the requirements for interoperability and to adopt common standards.

The consensus over a standard usually encompasses both the technical content of specifications and the legal conditions for their use. The conditions that the participants in standard-setting organizations (SSOs) are expected to agree are often codified in the internal rules (so-called "policies").[163] Ultimately, the conditions are expressed in licenses granted to copyrights, trade secrets, patents, *etc.*, related to the standard by their respective holders. The conditions may be generally divided into (1) royalty-free conditions (RF), and (2) reasonable and non-discriminatory conditions (RAND). Under RF, participants of SSOs are required to license their technology related to the standard on a royalty-free basis, whereas under RAND they are only required to impose reasonable and non-discriminatory license terms. The exact meaning of RF and RAND varies from SSO to SSO.[164] Many SSOs avoid making the meaning precise and leave it for the parties to negotiate. As a result, what constitutes RAND for some parties can be unacceptable for others.

Given the above and given the definition of an open standard as used in this thesis, only a situation where there are no material conditions on the exercise of the freedoms may lead to open standards. Many SSOs do not guarantee that a standard may be used in a Free Software program, since they might allow to impose restrictions incompatible with the freedoms. Even some major organizations allow participants to impose such restric-

162 Some parties cooperate by organizing so-called "plug fests" or exchange know-how using other means, without engaging formally with SSOs. Standards adopted in such a way are usually not open, but they may serve as a basis for the discussion in an SSO that could lead to making them open standards.

163 There exist various bodies that deal with standard-setting including "official" organizations with government backup or participation, such as ISO, more informal consortia such as W3C. It is also possible to treat as standard-setters organizations that manage patents included in patent pools (*e.g.*, MPEG LA). Here, we refer to all of them as SSOs, since all of them involve a degree of cooperation in standard setting that affects the openness of standards.

164 RF is differently understood in various SSOs. It is often allowed by SSOs for patent holders that follow RF to impose or agree on certain licensing conditions. See: WIPO (Standing Committee on the Law of Patents), *Standards and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf (Chapter IV discussing policies of various SSOs).

tions,[165] or they do not prescribe any requirements at all.[166] Additionally, any such requirements follow from internal rules of the organization only.[167] As such they may not bind third parties (non-participants), and they may even be hardly enforceable against the participants in a SSO.[168]

Here, we may conclude that standard setting is by itself not a sufficient guarantee that the resulting standards are open and that they may be used in Free Software. The necessary condition is that the standard setting procedure is properly organized. In particular, it has to lead to the designing of useful and complete specifications that fully conform to the definition of an open standard. We identify the following rule that follows from open standards (RU17).

---

165 The most prominent example is probably the common policy of ISO/IEC/ITU (see: http://www.iso.org/patents). Perens (2006) calls for elaborating such policies by the adoption of obligations to publish a reference implementation for any extensions to the standard (Bruce Perens, *Open Standards Principles and Practice*, at: http://perens.com/OpenStandards/Definition.html, referring to Sun Industry Standards Source License as an example). See also: *Sun Industry Standards Source License*, v.1.1., Sec. 3.1, at: http://www. OpenOffice .org/licenses/sissl_license.html ("In the event that the Modifications do not [comply with the standard], You agree to publish either (i) any deviation from the Standards protocol resulting from implementation of Your Modifications and a reference implementation of Your Modifications or (ii) Your Modifications in Source Code form, and to make any such deviation and reference implementation or Modifications available to all third parties under the same terms as this license on a royalty free basis...").

166 As far as we are aware, Polish Standardization Committee (PKN) does not have any policy with regard to copyright or patent rights (apart from requiring the transfer of all copyrights to translations of standards prepared by its members).

167 Albeit internal, they may still be subject to external review. For example in 2005 the Commission closed an antitrust investigation of the ETSI policy, after they had been amended to remove the risk of „patent ambushes" (The Register, *EC acts on patent ambushes*, at: http://www.theregister.co.uk/2005/12/14/patent_ambush/).

168 See, *e.g.*, *Rambus Inc. v. Infineon Technologies Holding North America Inc.*, 318 F.3d 1081 (C.A.Fed. Va. 2003) (Analyzing a policy of a standard-setting organization and holding that it contained a duty to disclose, but that it extended only to claims in patents or applications that „reasonably might be necessary to practice the standards" (at 1100). The duty did not cover „improvement patents, implementation patents, and patents directed to the testing of standard- compliant devices" (at 1101). The court did not find the policy to be clear enough to constitute the basis for a fraud verdict (at 1102). The dissenting opinion construes the scope of the duty to disclose to cover „patents and pending applications that might be involved in the work of the committee" (at. 1115)). See also: *Symbol Technologies Inc. v. Proxim Inc.*, 2004 WL 17701290 (D.Del.) (Admitting that laches is a defense to a patent infringement suit (at 3), but refusing it due to lack of proof that the patent holder had knowledge of infringing activity, (4-5) as well as the lack of requisite prejudice (5-7). The court did not find satisfactory reasons to allow equitable estoppel defense as well. The defendant claimed that the plaintiff was under an obligation to disclose any patents related to the standard elaborated by the organization to which both parties were members. However, the organization's bylaws allowed for non-disclosure with a statement that a license will be made available under RAND terms (7-9)).

Open standards: Users of a program are *allowed* to exercise (and *can* exercise) all six activities covered by the FSD with regard to the program, as long as the program uses an open standard.
In particular, developers are *allowed* to (and *can*) develop Free Software that uses such a standard, and distributors are *allowed* to (and *can*) distribute this software.

This rule is in relation to closed standards (RE25). Namely, open standards and closed standards exclude each other in the sense that a particular standard can be either open or closed (although a combination of different standards could be employed in a computer program). Also, the rule of open standards is in a direct relation with *the grant of freedoms* (RE26). Namely, if the program in question uses an open standard then (*ceteris paribus*) copyright holders are not stimulated not to grant users their freedoms. Additionally, third parties cannot restrict the freedoms if they are granted to a program that uses open standards (*ceteris paribus*).

The rule of open standards affects all four combinations presented in Table 3.1. Namely, it removes the limitations and restrictions that are the result of closed standards regardless of from whom the program is obtained and regardless of whether the program is original or improved Free Software. However, this is the case only to the extent that the program uses (or is developed in an attempt to use) open standards. In other cases, the rule does not apply. This means that for an efficient regulation using that rule, it is not sufficient that a relevant open standard exists, it is also necessary that it becomes popular and is actually used in programs.

We include in the framework:

(1) open standards (RU17),
(2) the relation between open standards and closed standards (RE25), and
(3) the relation between open standards and the grant of freedoms (RE26).

(3) Essential facilities

There are situations when designers of protocols, interfaces, or data formats fail to cooperate in the setting of an open standard.[169] For example, in a recent CFI case *Microsoft v. the Commission*, one of the issues was the extent of the right of a dominant company to restrict access to protocols used in its

169 Protocols, interfaces, and data formats may be developed individually by dominant market players, or collaboratively by multiple participants associated in a standard-setting organization. Under some circumstances, market dominants may develop their standards as open. These circumstances are unlikely to occur because of switching costs and network effects, as well as the inefficiencies created by non-market entities involved in the process. (See: Nicholas Economides, *The Economics of Networks*, 4 International Journal of Industrial Organization 673 (1996) at: http://www.stern.nyu.edu/networks/94-24.pdf; S.J. Liebowitz, Stephen E. Margolis, *Network Externalities (Effects)*, at: http://www.utdallas.edu/~liebowit/palgrave/network.html.) Development of open

software, *i.e.*, the right not to reveal the specification and to keep the protocols a closed standard. A question put before the court was whether the protocols constituted an *essential facility*[170] *for competitors. The Commission claimed that refusing access to interoperability information (i.e.*, specification of the protocols) constituted an abuse of Microsoft's dominant position and therefore the Commission demanded to make the protocols available under RAND conditions. The CFI upheld the Commission's decision, and the Commission agreed on the method of compliance proposed by Microsoft.[171] However, it has been observed that the particular conditions initially offered by Microsoft still restricted Free Software developers in their efforts to use the specifications in question.[172] Even the conditions offered at a later time seem to be still encumbered with restrictions that do not make the standards in question open.[173]

Here, we may conclude that the doctrine of essential facilities is again not a sufficient guarantee that the standards made available after the application of the doctrine are open and may be used in Free Software. First, the doctrine can be applied only towards dominants that abuse their market power. Second, license conditions imposed on such dominants might still discriminate Free Software developers.[174] Third, the effectiveness of essential facilities doctrine is further limited since competition law procedures are

standards by individual market players is unlikely especially if there is no perfect competition in the market. See: Stephen E. Margolis, S.J. Liebowitz, *Path dependence*, at: http://www.utdallas.edu/~liebowit/palgrave/palpd.html. A natural market process leading to open standards may be observed in the Internet. See: Mark A. Lemley, *Antitrust and the Internet Standardization Problem*, 28 Connecticut Law Review 1041, 1046 ("Indeed, the Internet is currently "owned" by a decentralized combination of [various entities] ... But the Internet can exist in this distributed condition only because each of the participants has agreed ... to a set of protocols that allows them to read and pass through messages sent by other participants."). However, open standards are more likely to be developed by standard-setting organizations that adhere to specific policies for this purpose (namely RF policies).

170 See: Wikipedia, *Essential Facilities Doctrine*, at: http://en.wikipedia.org/wiki/Essential_facilities.

171 Case T-201/04.

172 Microsoft offered conditions satisfactory to Free Software developers (namely Samba) only later on. See, *e.g.*, Andrew Bartlett, *A year since Microsoft's appeal failed*, at: http://people.samba.org/people/abartlet/a-year-since-microsofts-appeal-failed.html.

173 Although Microsoft made a great number of specifications available through its WSPP (Microsoft Work Group Server Protocol Program) and MCPP (Microsoft Communication Protocol Program), these standards are not maintained by an independent organization in an open decision-making procedure.

174 For example, the Commission required Microsoft to grant access to the specifications under RAND terms, which are generally not acceptable from the point of view of the FSD which does not allow to impose payment or any other material conditions as a consideration for *the grant of freedoms*.

time consuming.[175] Thus, essential facilities doctrine does not affect the rules identified so far, that restrict access to standards. Consequently, we do not find any rule that follows from essential facilities that should be included in the framework.

3.2.3 Conclusion on rules for standards and relations between them

In the preceding two subsections, we identified rules that limit or allow to restrict access to standards, as well as rules that regulate access to standards. We also identified relations between the rules, and the relations between them and other rules included in the framework so far. We present these rules and the relations in Figure 3.6.

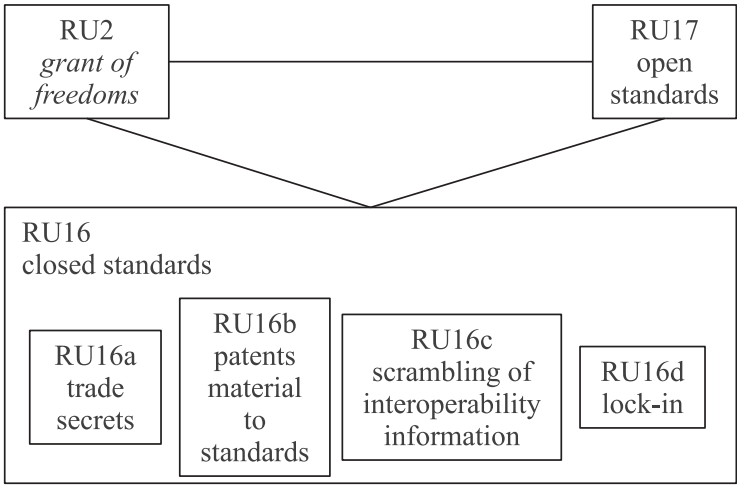

Figure 3.6: Rules for standards and relations between them

Here, we may conclude that we identified rules related to the standards scene and relations between these rules (as well as the relations between them and the rules related to the software scene). We include the rules and relations in our model of the current framework.

175 Remarkably, Microsoft v. the Commission case took roughly 5 years. See also: James Turney, *Defining the Limits of the EU Essential Facilities Doctrine on Intellectual Property Rights: The Primacy of Securing Optimal Innovation*, 2 Northwestern Journal of Technology and Intellectual Property 179 (2005). A similar case concerning IBM in the 80s resulted in the company's undertakings accepted by the Commission in lieu of an official decision. IBM pledged to disclose certain interface information for its hardware and software. See: F.M. Scherer, *Microsoft and IBM in Europe*, 2090 Antitrust & Trade Regulation Report 65, 66 (2003).

3.3 Reconstruction of a model of the current framework

In this section we reconstruct a model of the current framework. The model includes all rules and relations between them identified in previous sections. For a clear picture, we present separately the rules that regulate software (Subsection 3.3.1) and the rules that regulate standards (Subsection 3.3.2). Then, we present the regulatory environment in which all these rules operate (Subsection 3.3.3). Finally, we present all rules and relations between them in Figure 3.9 (Subsection 3.3.4).

3.3.1 Rules that regulate software

The basic rule that we included in the framework is (RU1) *the default rule*. Under *the default rule*, software is subject to exclusive copyrights which may be licensed or subject to a contract. The rule allows for private ordering using Free Software licenses. The following rule in the framework, based on *the default rule* is (RU2) *the grant of freedoms*. The freedoms are granted in Free Software licenses that allow users to undertake all six activities covered in the FSD (to run, study, adapt, modify, improve, redistribute, and release programs). Free Software licenses are also the source of other rules that we included in the framework. Namely, these are (RU3) *the right to fork*, (RU4) *copyleft*, and (RU5) *the hacker immunity*. *The right to fork* and *copyleft* attempt to provide the basic protection of the freedoms. They attempt to turn the freedoms from *inter partes* rights into negative and positive freedoms (respectively). In turn, *hacker immunity* attempts to provide for the basic incentive mechanism to developers and distributors of Free Software. Namely, it attempts to remove the most important demotivator, which is liability.

The remaining rules for software that we included in the framework are rules that limit or restrict access to software. After a thorough analysis, we included in the model of the framework the rules that follow from: (RU6) license proliferation and incompatibilities, (RU7) license revocability, (RU8) *inter partes* nature of licenses, (RU9) software-related patents, (RU10) contracts with distributors, (RU11) liability rules, and (RU12-15) non-legal regulators.

3.3.2 Rules that regulate standards

We have identified two rules for standards that are important from the point of view of the reconstruction of the model of the framework. These are (RU16) rule of closed standards, and (RU17) rule of open standards. After the analysis of issues related to closed standards, we identified four specific rules of closed standards included in the general RU16: (RU16a) trade secrets, (RU16b) patents, (RU16c) scrambling of interoperability information, and (RU16d) lock-ins. Additionally, after the analysis of issues related to open standards, we identified the rule of open standards (RU17).

Thus, we included in the model of the framework additional rules: (RU16) closed standards (with refinements), and (RU17) open standards.

3.3.3 Regulatory environment

The reader may attain an impression that under the model of the framework the relations between the licensor of a given Free Software program and all its users are the same from the legal point of view. In particular, the rights and obligations of the parties that result from these relations appear as *uniform.* We illustrate this in Figure 3.7.

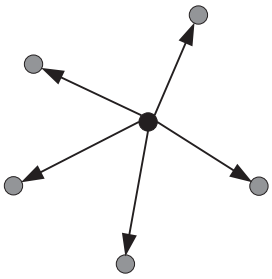

Figure 3.7: Uniformity

In Figure 3.7 solid lines represent relationships between the licensor (black dot) and users (grey dots). We draw all lines in the same pattern, in order to illustrate that all licensor-user relationships appear as *uniform*.

However, the uniformity should not be automatically assumed. Despite the adjustment, all legal rules in the framework operate in a regulatory environment. This environment is the result of national laws (the applicable laws). Notwithstanding some harmonization that has happened over the years, these laws are different across various jurisdictions. It means that there is no single set of legal provisions in which the licenses operate. Rather, in each particular case (*e.g.*, to every particular user of Free Software) different laws may be applied. The applicable law would be indicated using the rules of private international law of the forum. Given the fact that users are scattered all over the world, there are multiple fora. So, it is impossible to indicate

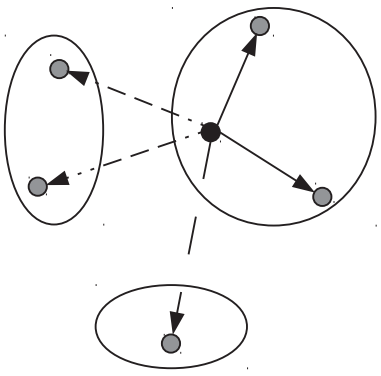

Figure 3.8: Regulatory environment

a priori one law that would apply. On the contrary, we should assume that users who are based in different jurisdictions are subject to different laws. The result is illustrated in Figure 3.8, with regard to relations between users and Free Software licensors (similar figures can be presented to illustrate relations between users and developers, distributors, patent holders, *etc.*).

In Figure 3.8, the black dot represents a Free Software licensor. The licensor is based in a certain jurisdiction, demarcated by solid circle. A pair of licensees (grey dots) is based in the same jurisdiction as the licensor, while three others are based in different jurisdictions. Although all relations with licensees are regulated using the same license, the final result of this regulation may vary because of the fact that different national laws apply to licensees based in different jurisdictions.[176]

It follows that there are as many regulatory environments, as there are many jurisdictions with different laws. We have to account for the differences between jurisdictions that materially affect user freedoms in the model of the regulatory framework.[177] We do not find any material differences on the general level. All national laws known to us authorize copyright holders to license their rights (or subject them to a contract). The exclusive copyrights and the freedom of contract which are the basis of the Free Software licenses have remained unquestioned, at least in jurisdictions that follow the rule of law. If under a particular national law Free Software licenses were invalid, this would lead to the conclusion that Free Software is used in breach of copyrights (certainly in case of developers and distributors, arguably also in case of all other users of this software). Such a conclusion would be hardly acceptable for anyone. In such a situation we expect that the law would be applied and interpreted in a way to find that the freedoms were effectively granted. So, we assume that national laws are the same to the extent they all generally allow Free Software licensing, and that the licenses are binding under all these laws.[178]

---

176 For example, under Polish Private International Law, when there is no agreement of the parties to the contrary, it would be the law of the residence, or of the seat, of the licensee (Janusz Barta and Ryszard Markiewicz, Oprogramowanie open source w świetle prawa, między własnością a wolnością [Open Source in the light of law, between property and freedom], 149 (Zakamycze 2005)). Free Software licenses usually do not contain choice of law clauses. The MPL is an exception here, since it points to the law of California, although in a somewhat vague way (most probably because it attempts to take into account the so-called "mandatory rules"): "*This License shall be governed by California law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions*" (See: http://www.mozilla.org/MPL/MPL-1.1.html).

177 An interesting source of information about differences in national laws is the IDABC document titled *Translation of EUPL v.1.0 into the official languages of the European Union - Report on comments received by IDABC*, at: http://ec.europa.eu/idabc/servlets/Doc?id=29987. It is the record of comments received from national experts in the course of translating the European Union Public License into 22 EU national languages.

178 *Cf.* FN 82. See also: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 153.

National laws differ materially on more specific issues. For example, some of them provide for certain requirements as to the validity of licenses by requiring a specific wording for a license to cover all six activities specified in the FSD effectively.[179] Some national laws also contain non-waivable copyright restrictions, such as moral rights[180] or non-waivable royalties.[181] They additionally allow to revoke a license if it fails to meet certain criteria or perhaps even to revoke any license at all.[182] Moreover, national laws contain different liability rules. Some national laws provide even for non-waivable liability.[183] In order to account for such specific differences in national laws we have to include in the framework a general rule that limits all legal rules in the framework. Importantly, this rule does not affect non-legal rules (the ones which follow from regulators other than the law). We call this rule the "regulatory environment" (RU18).

> **Regulatory environment** : All legal rules in the framework are subject to the applicable national law.

Since there are fourteen legal rules that we included in the framework so far, there are fourteen relations between this rule and the legal rules (RE27-39).

3.3.4 Graphical presentation of the model of the current framework

In Figure 3.9 we present the model of the current framework reconstructed in this chapter.

---

179 Under Polish copyright law a license covers only such "fields of endeavour" which are specified in the license. A program cannot be used by the licensee on other fields. Also, the term of a license is 5 years unless the parties decide otherwise (*i.e.*, unless the license was expressly granted for an unlimited period of time). Additionally, if the license is silent on the covered territory, the licensee may use the program in Poland only.

180 See *e.g.*, Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 122.

181 Under Polish copyright law moral rights have been significantly limited with regard to computer programs. Still, an author of a program has an non-waivable right of authorship and attribution. Other jurisdictions could provide for a broader (and non-waivable) protection of moral rights, for example against certain modifications or uses of a program, while under Free Software licenses such activities are not restricted. Also, under Polish copyright law the royalties from private copying levies are non-waivable, while under Free software licenses the licensee is not required to pay any royalties.

182 Under Polish Copyright Act, a license granted for unspecified period of time can be terminated by either party. Non-revocability of licenses is a condition of FSD-compliance (see Subsection 2.1.2). This means that licenses found FSD-compliant in some jurisdictions could fail compliance test in some other jurisdictions (*e.g.*, in Poland). However, the FSF has not undertaken such a test across jurisdictions so far. Still, the differences as to the revocability of licenses do affect user freedoms.

183 Under Polish law, it is not possible to waive liability for damages caused wilfully. Additionally, consumer protection laws prohibit waiving liability in consumer relations.

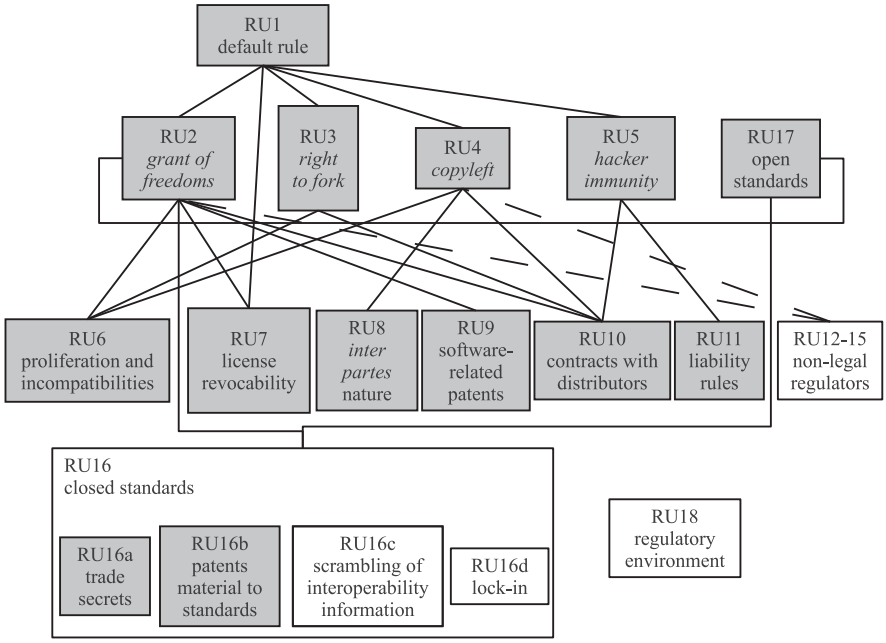

Figure 3.9: Graphical presentation of the model of the current framework

Figure 3.9 contains rules and relations between them as already presented in Figure 3.5. In Figure 3.9 we additionally include rules for standards and relations between them. We note that we do not present all 39 relations between the rules that we identified throughout the analysis; we present only the relations that we find most relevant for the purpose of our study. We also include the regulatory environment rule. This rule affects all legal rules in the framework. It does not affect the non-legal regulators. For the sake of clarity, instead of representing all 14 relations between the regulatory environment and other rules with lines, we mark them by filling the boxes that represent the rules affected by the regulatory environment in light grey. The boxes of non-legal rules (and of the regulatory environment itself), which are not affected, are white.

In Figure 3.9 we continue to present the rules grouped in three levels, as in Figure 3.5. On the first level there is *the default rule*. This is the rule which empowers copyright holders to grant user freedoms or to refuse to grant them. On the second level there are 4 rules of software that result in the grant of user freedoms or in the protection of the freedoms (*the grant of freedoms*, *the right to fork*, *copyleft*, and *the hacker immunity*). These rules were already presented on that level in Figure 3.5. In Figure 3.9 we amend the second level with one more rule that protects user freedoms. This is the rule of open standards. On the third level there are 10 rules that constitute limitations or restrictions of user freedoms, as it was the case in Figure 3.5. In Figure 3.9 we amend the third level with rules of closed standards and with the regulatory environment rule.

We note that the third level currently contains the following items: (1) license proliferation and incompatibilities, (2) license revocability, (3) the *inter partes* nature of licenses, (4) software-related patents, (5) contracts with distributors, (6) liability rules, (7) non-legal regulators (a class that includes 4 non-legal rules of software), (8) closed standards (a rule that consists of 4 rules of closed standards), and (9) the regulatory environment. We will refer to all these nine items as "limitations and restrictions of the freedoms" (in Chapter 6 we will refer to them jointly as the "inefficiencies of the current framework").

3.4 Chapter conclusions

In this chapter we reconstructed a model of the current regulatory framework of Free Software. The framework consists of (1) rules that regulate the freedoms and (2) relations between the rules. The rules included in the model regulate both (1) access to software and (2) access to standards. They regulate user freedoms in all four combinations presented in Table 3.1.

The framework includes the rules that grant and attempt to protect user freedoms, as well as the rules that limit or restrict the freedoms. Many of the limitations and restrictions are not sufficiently addressed by the rules that we included in the model. It follows from the model that the freedoms are not sufficiently protected under the current framework.

Thus, we shall proceed to analyse the relations between software communities and eGovernments in order to identify how they affect the framework in the protection of user freedoms. If after the analysis we should still find no adequate protection, we shall design necessary improvements of the current framework and propose an improved framework.

4 Communitarian protection of user freedoms

In this chapter we focus on software communities and on how they affect the model framework reconstructed in Chapter 3. We identify relations between software communities and user freedoms. Below, we recall our findings so far and then stipulate the working programme.

In Chapter 1 we found that conditions of user freedoms are the same as the conditions of the control over the working of the programs. They are: (1) access to source codes and (2) access to specifications of standards. Only given the access, users are able to exercise their freedoms, that is to perform all six activities covered by the FSD: (1) to run, (2) to study, (3) to adapt, (4) to improve, (5) to release, and (6) to redistribute programs. In Chapter 2 we found in particular that in order to allow users to exercise all these activities, it is necessary that copyright holders allow users to exercise all activities covered by copyright. In Chapter 3 we explained in particular that copyright holders that follow the Free Software approach do so by granting to users licenses that comply with the FSD – the Free Software licenses.

In Chapter 3 we established that under the licenses users are allowed not only to use, but also to develop and distribute Free Software. The rules that are a result of Free Software licenses (*i.e.*, *the grant of freedoms*, *the right to fork*, *copyleft*, and *the hacker immunity*) together with other rules that regulate the use, development, and distribution of Free Software, and relations between the rules form the regulatory framework. Moreover, we reconstructed a model of the framework. In the model, we included also the rules that constitute limitations and restrictions of user freedoms. *Ceteris paribus*, given these limitations and restrictions users cannot exercise their freedoms in an undisturbed manner. It follows that user freedoms are not sufficiently protected under the model.

After this summary of findings, we emphasize the course of research activities culminating in an attempt to answer the Problem Statement 1 (PS1). In Chapter 1 we found that software communities play an important role on the Free Software scene. Within the communities, the participants are guided and supported in their efforts to use, develop, and distribute Free Software. Precisely speaking, in this thesis (see Chapter 2) we define a software community as a group of entities that collaborate in the development of a Free Software project and that may also distribute the project to other users, as well as that provide guidance on its use. Additionally, we stated that in our research we focus on communities of hierarchical and quite formalized structures because we anticipate that such communities have the strongest possible impact on user freedoms. In Chapter 3 we formulated an assump-

Towards_an_Improved_def.indd 117 23-02-2010 10:49:25

tion that communities enter into relations with the freedoms.[184] In this chapter we identify these relations. We do it in order to answer the PS1 and RQ1 of this thesis, which we repeat below.

PS 1: *What are the relations between user freedoms and software communities?*

We answer the Problem Statement 1 by answering the Research Question 1, which is:

RQ 1: *In what way do software communities affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?*

In order to answer the RQ1, we start by elaborating on limitations and restrictions in the exercise of the freedoms undertaken by users individually (Section 4.1). Then, we elaborate on the exercise of the freedoms as organized in software communities (Section 4.2). In the end, we present chapter conclusions, an answer to the RQ1, and an answer to the PS1 (Section 4.3).

4.1 Limitations and restrictions of the freedoms

All programs usually have bugs, lack all necessary features, and are not perfectly interoperable with other programs. So, the development and distribution of programs have to be organized and repeated whenever new bugs are found and fixed, new features are requested and added, or interoperability is broken and provided for again. Otherwise, the programs will not work properly. Repeated and organized development and distribution are usually referred to as "maintenance". Unless software is maintained it quickly deteriorates in one way or another. Conversely, if software is maintained, it means that at any given point of time it is possible to fix bugs, include new features, and provide means for interoperability effectively. Maintenance is a means for making sure that software works properly. In other words, maintainers control the working of programs. Theoretically, it is possible to control the working of a program that has not been maintained for some time, but this requires much more efforts and usually proves impracticable.

User freedoms granted in the Free Software licenses allow users to maintain programs on their own (generally, maintenance means that all six activities covered in the FSD are undertaken in an organized and continuous process). On average, however, users do not have skills and resources (hardware, development tools, data, know-how, *etc.*) necessary to maintain programs

184 See FN 103.

effectively.[185] Usually, only qualified engineers (*e.g.*, hackers) or specialized firms possess some or all such skills and resources. Average users are not able to maintain programs effectively even if they have the rights to do so. Such users have to have their programs maintained for them by someone else. This does not mean, however, that average users are unable to control the working of programs. If they are allowed to exercise all six activities covered in the FSD, they can *outsource* the maintenance to persons of their choice.[186] As a result, they control the working of programs through such outsourced maintainers. It follows that as long as users are allowed to exercise all six activities covered in the FSD they can control the working of programs even if they personally lack the necessary skills and resources. What is essential here is not who has the skills and resources, but whether users can *freely outsource* the maintenance of software.

Let us assume a hypothetical situation that there is no community around a Free Software program. In such a situation, users of the program can still attempt to exercise their freedoms granted in the Free Software license of the program individually. By *individual exercise of the freedoms* we mean either of two scenarios: (1) users become actors on the software scene by maintaining Free Software themselves, or (2) users outsource the maintenance to a person of their choice.[187] In both of these scenarios users perform some or all of the six activities covered in the FSD (or benefit from the performance of these activities by persons to whom the maintenance is outsourced). However, users do not enter into any complex relations with other users in maintaining software. In particular, users do not form a software community.

The individual exercise of the freedoms is limited and can be further restricted in a relatively easy way. This is due to the rules that limit or allow to restrict the freedoms that we already identified and included in the model of the framework reconstructed in Chapter 3. Below, we identify *how* these limitations and restrictions affect the individual exercise of the freedoms exactly, by drawing from our findings presented in Chapter 3. We address them in the following order: (1) license proliferation and incompatibilities,

185 This holds true not only in case of individuals, but also in case of corporate users, such as firms, who nevertheless have to outsource maintenance of software due to various considerations.

186 To put it in another way: If users who possess all necessary skills and resources are allowed to exercise the freedoms without limitations and restrictions, there is nothing that legally prevents them from maintaining the programs for average users.

187 These two scenarios are to a large extent two sides of the same coin. Namely, users who maintain software themselves usually do it for other users who outsource the maintenance. Certainly, there are users who maintain software for themselves only. Here, under the heading of "users who maintain software themselves" we include both "users who maintain software for other users" and "users who maintain software for themselves only". Needless to say, from the point of an average user, the ability of the former users to exercise the freedoms is most important.

(2) license revocability, (3) *inter partes* nature of licenses, (4) software-related patents, (5) contracts with distributors, (6) liability rules, (7) non-legal regulators of software, (8) closed standards, and (9) regulatory environment. After discussing all of them we present (10) conclusion on limitations and restrictions in the individual exercise of the freedoms.

4.1.1 License proliferation and incompatibilities

The proliferation and incompatibilities of Free Software licenses materially affect user freedoms. Certainly, there are tendencies that result in a gradual decrease of the number of model licenses used in practice, as well as tendencies that allow to overcome incompatibilities (such as dual licensing, amending licenses with compatibility clauses). However, despite these tendencies it is still the case that any user who wants to exercise the freedoms individually in a *combination* of selected programs has to undertake a two-step legal analysis. First, the user has to determine whether each license of programs included in the combination allows to exercise all six activities covered in the FSD. Second, the user has to determine whether the licenses allow to exercise them with regard to the combination as a whole.[188]

In the first step of the analysis the user can benefit from the work already performed by the FSF. Here, we should recall that the FSF scrutinizes the compliance of various licenses with the FSD and publishes an official list of the analysed licenses. Licenses found by the FSF as FSD-compliant are also commonly recognized[189] as allowing to exercise all six activities covered in the FSD. It follows that any user can attempt to mitigate the effects of license proliferation by exercising the freedoms only in the software that is available under FSD-compliant licenses. Still, the user has to find programs under these licenses, which is not necessarily an easy task. It involves searching through various sources which can differ extremely both as to the type and quality of software they offer, as well as to the way they present their licensing terms. There is no single point of reference where users could browse for FSD-compliant software, but there exist some popular public repositories.[190] Using such repositories can be helpful, since they offer useful querying interfaces. Some of them, for example, allow to query the database of hosted software using license as a criterion, but naturally such services are not pro-

188 For an excellent description of the procedure of such an audit see Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 53 *et. seq.* as well as 71 *et. seq.*

189 Such common recognition may be helpful since Free Software licenses are model licenses. The exact scope of user rights and obligations depends on how a model license is applied between parties to a transaction. Common recognition is a custom or trade practice, which are legally relevant in determining how the license is applied between the parties.

190 Examples include SourceForge (http://sourceforge.net), FreshMeat (http://freshmeat.net), or (last but not least) the Free Software Directory http://directory.fsf.org/.

vided under any legal guarantee.[191] At the end of the day, the user has to review each program and its license separately, and seek for legal advice if necessary.[192]

The second step of the analysis is performed on programs under licenses identified in the first step. Given our findings in Chapter 3, it is legally impossible to distribute programs under incompatible licenses in some combinations. So, the second step comprises identification of incompatibilities between these licenses. If the identified incompatibilities cannot be resolved (*e.g.*, by combining programs in a way that does not trigger incompatible obligations), the second step comprises also the removal of respective programs from the combination.[193] Notably, the second-step analysis has to be performed only by users who want to distribute the combinations (*i.e.*, those who intend to maintain them for other users). Users who maintain software only for themselves can proceed immediately after step one. This means that the results of the second step materially affect average users, *i.e.*, the ones who do not maintain software themselves and outsource maintenance. Without the completion of the second step, provision of maintenance to such users might constitute license infringement.[194]

In the second step, similarly to step one, a *prima facie* identification of incompatibilities can be found at the FSF's webpage. Namely, the FSF does not only analyse licenses for their FSD-compliance, but also they indicate whether a given license is GPL-compatible. This means that all users have a point of reference, at least as far as GPL-compatibility is concerned.[195] Some additional information can be deducted by identifying what software is distributed by major Free Software distributors and the manner they distribute it (in which combinations, *etc.*). Users could assume with some degree of risk[196] that these distributors comply with all licenses of the hundreds of

191 From the formal point of view, such a query is not sufficient to determine that a particular license was granted. This might become important if under the applicable law mere use of Free Software is insufficient for concluding a valid license (for example because license terms were not communicated to the user clearly enough).

192 Even FSD- and OSD-compliant licenses, still include obligations for the licensees, which have to be complied with.

193 The removed programs have to be maintained separately, or at least not distributed together with the combination.

194 Most probably, mere outsourcing of the maintenance of software would not constitute distribution by a user, or at least not to the extent necessary to find a material license infringement. However, provision of maintenance services for such a user might involve distribution (*e.g.*, of improvements of software). In such a situation the distributor has to identify applicable obligations and resolve any incompatibilities.

195 Even the FSF's opinions are not legally binding and are not provided with any legal guarantee. Additionally, what matters is how the license is interpreted in a particular transaction.

196 Actually, this risk can be considerably high, because distributors could proceed according to individual arrangements with particular licensors, not on the basis of publicly granted model licenses.

Towards_an_Improved_def.indd 121 23-02-2010 10:49:26

Free Software programs they distribute, or at least refer to the conduct of the distributors as an established market practice.[197]

Still, even if a user exercises the freedoms only in the combinations of programs released under compatible licenses, the user has to comply with all obligations specified in the licenses of all programs in a combination. In order to do so, the user has to identify these obligations. A brief analysis of the most popular Free Software licenses reveals that they usually require providing certain information to users to whom the programs are distributed (preserving copyright notices, passing through license texts, communicating warranty disclaimers and liability limitations, *etc.*). Also, given the popularity of *copyleft* licenses, there is a high probability that at least some of the licenses of the programs in the combination require making source codes available,[198] if the programs are distributed.[199] Since there is a dozen of FSD-compliant licenses, the complete list of exact obligations can be quite long. However, in any given combination of Free Software programs there is a definite number of such obligations. So, the completion of the second step of the analysis is not impossible, but may prove impractical for some users.

Here, we may conclude that license proliferation and incompatibilities stimulate some users not to exercise the freedoms individually. They especially stimulate some users not to maintain certain combinations of software for other users. The more programs in a combination and the more complex the resulting license relations are, the fewer users are willing to maintain it. The stimulation does not materially affect users who possess skills and resources necessary to maintain the software themselves, as well as those who can afford completing both steps of the legal analysis. So, most average users who want to control the working of Free Software do not have as wide a choice of maintainers as they would have if there was no license proliferation and incompatibilities. Also, due to costs of the legal analysis, the maintainers are not as effective in developing and distributing software as they could be. Certainly, at least some resources that they could direct to maintenance has to be directed to legal analysis.

4.1.2 License revocability

License revocability allows to restrict user freedoms. In fact, it leads to a complete removal of the freedoms. We found in particular that even a significant

197 See FN 189.

198 Depending on the scope of the respective *copyleft* clause and the facts of the particular case, this obligation can apply to the program itself, to the whole combination, or to something in-between. We are unaware of a *copyleft* clause that would require to make available source codes of all imaginable combinations. Even the *copyleft* clause of the GPL seems to be limited to derivative works and "mere aggregations" are explicitly excluded from its scope.

199 Usually, users who do not distribute the software but only use and develop it internally are not bound by these obligations (even Affero GPL or Open Software License are concerned with making software available to third parties, not internal use).

Towards_an_Improved_def.indd 122 23-02-2010 10:49:26

user base or a developed downstream distribution does not make it impossible to unilaterally terminate the license in practice, if this is allowed by the applicable law. After the licensor unilaterally revokes a Free Software license effectively or otherwise terminates the legal relationship with users, then *the default rule* of exclusive control over software is reinstated. Also, in such a case users cannot rely on *the right to fork* or *copyleft* to defend their freedoms, as both of them require a binding license to exist. More importantly, they do not have any freedoms to defend, as only the terminated license constituted their source.

Here, we may conclude that if the license terminates, then from the legal point of view users cannot exercise their freedoms in any way, also not individually. From the practical point of view, the licensor would probably not take legal steps against users who continued to use and maintain the software for themselves only, without distributing it. More importantly, however, users could not legally maintain software for other users. All users would be precluded from the control over the working of the programs, maybe apart from a few.

4.1.3 *Inter partes* nature of licenses

Free Software licenses form *inter partes* relations between licensors (copyright holders) and licensees (users). As a consequence, unless there is a legal rule to the contrary, third parties (*i.e.*, entities other than the licensors) cannot easily enforce *copyleft* clauses against the licensees. Specifically, if there are no rules in the applicable law that allow third parties to benefit from an *inter partes* relation effectively, the enforcement of *copyleft* remains in the hands of the licensors only. We illustrate this in Figure 4.1.

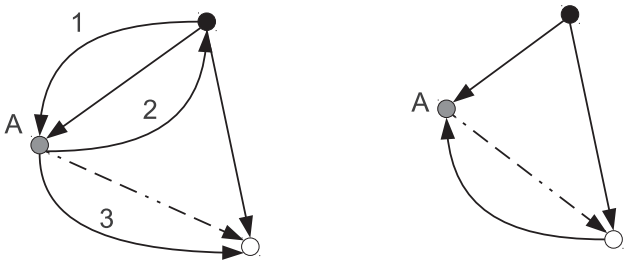

Figure 4.1: Copyleft

In Figure 4.1 we present two example sets of relations under *copyleft*. On the left of Figure 4.1 we see user A who creates an improvement of an original Free Software program but does not offer it to other users under a Free Software license. Specifically, the terms of user A contain restrictions that prevent users from exercising their freedoms individually in the improvement. This is represented by the dashed line as opposed to the solid lines of relations based on the Free Software license to the original. Thus, users may not exer-

cise their freedoms individually in the improvement. However, as the original is subject to a *copyleft* clause, the licensor of the original may legally enforce that the improvement is released under a Free Software license, and all its users allowed to exercise the freedoms in it. The enforcement relation is marked by 1. Then, depending on the exact legal structure of the particular *copyleft* clause, user A grants the freedoms to the improvement to the original licensor (relation marked by 2), or user A grants the freedoms directly to other users (relation marked by 3). On the right of Figure 4.1 we illustrate an even simpler use of *copyleft*. Therein, the original licensor does not interfere. Other users address their claims based on *copyleft* directly to user A. This is possible if only the applicable law empowers the users to enforce the *inter partes* relation between user A towards the licensor. If not, then the users have to follow the scenario presented on the left of Figure 4.1.

Obviously, the enforcement requires the enforcing person's resources invested in gathering evidence, communicating with the infringing party, obtaining legal advice, and supporting the case before a court.[200] As long as the person is an average user that does not have the necessary resources, the ability to enforce *copyleft* is effectively limited, even if the applicable law allows such a person to enforce *copyleft*. This means that the *inter partes* nature of the licenses limits users in exercising their freedoms individually. Namely, in case a user wanted to exercise their freedoms in Free Software programs distributed in breach of a *copyleft* clause, the user would have to invest considerable resources in enforcing the clause, or even have to rely on the licensor to enforce the *copyleft* (if the applicable law restricted third party standing). Until the *copyleft* was successfully enforced, the user could not maintain, or outsource the maintenance of the program "appropriated" by user A.

Certainly, the user could avoid the above-mentioned limitation by exercising the following sequence: (1) obtaining the program in question directly from the licensor, (2) exercising *the right to fork*, and (3) creating a substitute improvement. We illustrate this in Figure 4.2.

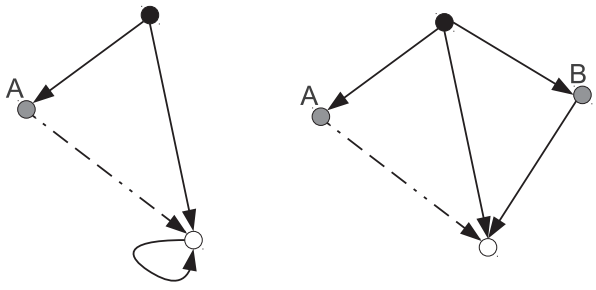

Figure 4.2: The right to fork

200 According to Eben Moglen most of the disputes related to Free Software licenses have been settled out-of-court (See: Eben Moglen, *Enforcing the GNU GPL*, at: http://www.gnu.org/philosophy/enforcing-gpl.html).

In Figure 4.2 we present two example sets of relations under *the right to fork*. On the left of Figure 4.2 we see the same initial situation as on the left of Figure 4.1: user A does not offer the improvement to other users under terms as permissive as the Free Software license of the original program. If the original is not subject to a *copyleft* clause, or if the enforcement of *copyleft* obligations is impossible or impractical, users can still resort to *the right to fork*. Under *the right to fork*, users may attempt to develop a substitute improvement individually and maintain it themselves. Alternatively, (see the right of Figure 4.2) another user (user B) could fork the program, improve it, and offer the improvement to other users under the terms as permissive as the original license. Notably, under *the right to fork*, users may not exercise their freedoms in the improvement developed by user A. Instead, they are only allowed to develop (or wait until the licensor or user B develops) substitute improvements on the basis of the original program. Definitely, developing and maintaining a substitute improvement requires more skills and resources than maintaining an already developed improvement of user A, made available in performance of *copyleft* obligations or out of free will of user A.

Here, we may conclude that the *inter partes* nature of licenses limits some users in becoming maintainers of certain improvements of software for other users. Similarly to other limiting and restricting rules described so far, it does not materially affect users who possess skills and resources necessary to maintain the software themselves, as well as those who can afford enforcing *copyleft* (provided that the applicable law does give them standing). So, it mostly affects average users, but other users can be also materially affected in specific circumstances.

4.1.4 Software-related patents

It is possible to restrict user freedoms using various rights that are not or may be not licensed by a Free Software licensor, such as patent rights. Despite the territorial limitation of patents, we did not find any major jurisdiction that would not allow to obtain patents in one way or another related to software. We found that patents can be used to restrict user freedoms even by a patent holder who is a third party completely unrelated to the parties of a Free Software license. As a result, parties to any such license cannot easily avoid the threat of patents, in particular by contracting this risk away between themselves.[201] If any party infringes a patent, the party may be liable towards the patent holder. More importantly, the patent holder could prevent the exercise of the freedoms, or subject it to conditions not compatible with the freedoms.

Usually, it is not easy to discover that there is a risk of patent infringement. The discovery is possible using patent searches, but the searches

201 As a matter of facts, such risk cannot be easily avoided by parties to a proprietary license as well.

involve browsing through large amounts of patent documentation. The documentation is not always easily accessible and complete.[202] It is also not easy to analyse, mostly because it does not usually disclose the actual source code implementation of software-related inventions, or at least the exact algorithm used in these inventions. This means that many patent infringements may be accidental, while their avoidance requires considerable skills and resources.

Here, we may conclude that software-related patents allow to restrict some users not only in becoming maintainers of certain software for other users (*i.e.*, such software that is found to infringe a patent), but even in a simple use of such software. Patents do not affect only such users who possess skills and resources necessary to avoid the risk of patent infringement or to work around the patent (assuming that this is legally and technically possible). So, software-related patents definitely affect average users, and it is likely that most other users are also materially affected by them.

4.1.5 Contracts with distributors

Contracts may be used to restrict user freedoms regardless of other limiting and restricting rules. Given the autonomy of the parties to a contract, there are practically no boundaries as to what the restrictions may be. Generally, it is possible and has been observed in practice that users accept contracts that prevent them from exercising their freedoms individually. From the point of view of the individual exercise of the freedoms, it is important that such contracts include contracts whereby (1) users undertake not to maintain software themselves, (2) users undertake not to outsource maintenance otherwise than to the other party of the contract, or (3) users undertake not to perform both of the above. Sometimes, users do not undertake anything, but the contract conditions the performance of warranties or services related to Free Software on the user not performing the maintenance. This alone may effectively prevent many users from maintenance.

Here, we may conclude that distributors can use contracts to restrict some users in practically any exercise of the freedoms. In particular, they could restrict users in becoming maintainers of certain software for other users (*e.g.*, the software as improved by the distributors). Certainly, such contracts do not affect users who do not accept them, but naturally average users are more likely to accept such contracts than users who possess skills and resources necessary to maintain the software themselves. At the same time, average users are most affected by such contracts, since as a result they cannot outsource the maintenance otherwise than to their contractors, while they would not maintain the software themselves anyway.

202 See: WIPO (Standing Committee on the Law of Patents), *Report on the International Patent System*, (3 February 2009, SCP/12/3 Rev. 2), p. 22 *et. seq.*

Towards_an_Improved_def.indd 126 23-02-2010 10:49:27

4.1.6 Liability rules

Liability rules existing in various national laws affect *the hacker immunity*. Here, we should recall that *the hacker immunity* is a broad warranty disclaimer and liability limitation clause. The immunity attempts to cover nearly all possible liability related to Free Software (warranties, contract or tort liability, sometimes also third party claims, *etc.*). This means that *the hacker immunity* attempts to pass to the user the risk related to software.

If *the hacker immunity* is fully enforceable, users are left in the same situation as users of proprietary software, as far as such a risk is concerned. Namely, unless they are able to repair the program or remedy the infringement of third party rights themselves, or unless distributors of software offer separate warranty or indemnification, users bear the risk of damages or liability. In such a case, the exercise of the freedoms could constitute a heavy economic burden for users. This means that the full enforceability of *the hacker immunity* stimulates users not to maintain software themselves. Users are stimulated to outsource maintenance to persons who can guarantee good quality, most preferably together with some kind of indemnification against third party rights. Consequently, the full enforceability of *the hacker immunity* limits users who cannot offer that quality and indemnification in their ability to offer maintenance services.

Conversely, if liability rules do not allow to enforce *the hacker immunity* in whole or in part, developers or distributors of Free Software might be required to pay damages or otherwise bear liability for defective Free Software programs. In such a case, the rule of *the hacker immunity* would fail to remove an important demotivator for prospective contributors to Free Software projects, including those who would be willing to maintain software on behalf of others. Only those who possess necessary resources would then be willing to undertake to maintain the software.

From the legal point of view, the relation between *the hacker immunity* and liability rules boils down to the question who will bear liability for defective software and reimburse damages. However, from the practical point of view it is often the case that the damages are not the main issue, since the operation of the user's enterprise or the safety of data, *etc.* is much more important. Then, it is crucial whether the user can rely on someone who not only guarantees certain level of service and is liable for not meeting the level, but also on someone who can be called upon in order to repair the defects and bring the software back to operation. Obviously, users who possess sufficient skills and resources can provide such a service themselves, if only they have access to source codes of the software and specifications of standards used in this software. For users who have to outsource maintenance, this is not sufficient.

It follows that the freedoms will be exercised in an unencumbered way if it is possible for users to maintain or outsource the maintenance of Free Software in a way that allows for a prompt removal of defects. It is less important that as a result of the relation between *the hacker immunity* and liability

rules the liability can or cannot be attributed to certain actors. It is more important whether there exist actors who provide maintenance services, and in particular offer to remove defects found in software.

4.1.7 Non-legal regulators of software

User freedoms may be restricted using non-legal regulators. In particular, it is possible to restrict user freedoms using a special architecture, *e.g.*, by making software a part of a service or a device. User freedoms are also limited by the way a market operates and norms are imposed. Market and norms affect mostly *copyleft*, but we also found a material limitation on the exercise of user freedoms in general through market regulation.

Here, we may conclude that from the point of view of the individual exercise of the freedoms it is important that all the identified non-legal regulators affect maintenance of software. The result of the architecture is similar to the result of contracts with distributors, but instead of legally obliging users to refrain from maintaining software on their own or from outsourcing it to a third party, the architecture makes it simply impossible or impractical. The market regulation can additionally limit or restrict users from maintaining software. Transaction costs make it costly to become an actor in the Free Software scene. The market also fails to stimulate users to perform their *copyleft* obligations in a way that would allow users to access source code of some programs and exercise their freedoms in practice. Finally, integration costs can prevent users from undertaking maintenance of software, even if this is legally and technically possible.

4.1.8 Closed standards

Trade secrets, patents material to standards, scrambling of interoperability information as well as locking-in lead to closed standards and affect user freedoms. Also, neither reverse engineering, nor standard setting as such, nor the doctrine of essential facilities is sufficient for removing the identified limitations and restrictions on standards and making them open standards. It is additionally necessary that the standard setting procedure is properly organized so that the resulting standards are open. It is also necessary that such open standards become popular and are actually used in software. Otherwise, the software will continue to be developed according to closed standards.

Here, we may conclude that due to closed standards, there are limitations and restrictions on the maintenance of software. Users who are not able to access standard specifications (or legally use information contained therein) are precluded from maintaining the software, at least as far as maintenance involves providing for interoperability using a closed standard. Depending on circumstances, they are precluded either from maintaining it for other users only, or from maintaining it for themselves as well.

Towards_an_Improved_def.indd 128 23-02-2010 10:49:27

4.1.9 Regulatory environment

The framework operates in a regulatory environment. In fact, there are as many regulatory environments, as there are many jurisdictions (national laws) involved in a particular case. As a result, we subjected all rules in the framework to the applicable national law. We did not find any laws that invalidate any of the rules included in the model of the framework as a whole. Still, we found that due to the differences between various national laws the rights and obligations that follow from these rules could differ across jurisdictions. For example, we can assume that there may be jurisdictions where despite *copyleft* clauses licensees would not be obliged to deliver some improvements as Free Software, or where the law would not allow to enforce this obligation as effectively as in other jurisdictions. Additionally, in some jurisdictions, developers and distributors may not be allowed to waive their liability for Free Software as much as in other jurisdictions.

Given that there is a number of regulatory environments, if a user attempted to maintain software for other users, such a user would start entering into numerous relations with other users. For example, the user would become the licensor with regard to the improvements made to the software. The user could also conclude contracts with other users (distribution contracts, maintenance contracts, *etc.*). Obviously, given the globalization and the use of the Internet in the trade, such other users would most probably come from many different jurisdictions. Thus, there is a high probability that a different national law could apply in each case. It follows that the user would have to verify the impact of each national law on all relations.[203] Not everyone can or is willing to cover costs of such a verification.

Hence, differences between national laws constitute a limitation on the individual exercise of the freedoms performed by maintaining software for other users that come from different jurisdictions. The differences only do not affect the users who possess skills and resources necessary to (a) verify the legality of their conduct in all jurisdictions involved, and (b) to work around any identified legal obstacles. So, it definitely affects average users, and it is likely that many other users are also materially affected by the differences.

4.1.10 Conclusion on limitations and restrictions of the freedoms

As follows from the above, the individual exercise of the freedoms may be subject to severe restrictions and limitations in practice. Despite *the grant of freedoms* in Free Software licenses and despite some rules that attempt to protect the freedoms, such as *the right to fork* and *copyleft*, users may not at all times freely use Free Software by undertaking some or all activities covered

203 Certainly, some of these problems could be avoided using choice of law clauses in the contracts with users (pointing to one jurisdiction only). However, there are some laws (so-called mandatory rules), the applicability of which cannot be avoided.

by the FSD. Thus, we may conclude that elimination of these restrictions and limitations is a necessary condition in order to allow users to exercise their freedoms effectively.

More precisely, in order to conclude that the software communities affect the framework by contributing to the protection of the freedoms, we have to establish whether the communities:

(1) decrease the number of Free Software licenses and minimize incompatibilities between them,
(2) effectively petrify the legal relationship between licensors and licensees,
(3) stimulate effective enforcement of *copyleft* or simply effective performance of *copyleft* obligations,
(4) prevent infringement of software-related patents or the use of such patents against Free Software users,
(5) prevent the use of contracts by distributors in order to restrict user freedoms,
(6) remove liability for Free Software while at the same time reduce the number of defects in Free Software (both technical and legal, if any),
(7) provide for proper organization of Free Software development,
(8) regulate standard-setting procedures in a way that the resulting standards are open standards and become popular, and
(9) minimize differences between national laws.

Conversely, if we do not find all of the above we will most probably conclude that the communities do not affect the framework positively or that they affect it to a detriment of user freedoms.

4.2 The freedoms in software communities

In the previous section we worked under the assumption that users attempt to exercise their freedoms individually. Under that assumption, while maintaining a program for themselves or even for other users, such users did not enter into any complex relations with other users. Let us change this assumption and assume that there are users who form a software community around the program and who maintain it as a Free Software *project* (the project may consist of the program alone or of a combination of many programs).[204] Briefly speaking, such a community consists of various entities that collaborate in the development of the project and that may also distribute the project to other users, as well as that provide guidance on its use.

204 For the explanation of "project" and "software community" see Section 2.4. See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 25.

In particular, in a community there are usually some (often quite numerous) participants that contribute to the project in many ways. Contributions include all kinds of improvements to programs incorporated in the project.[205] The resulting copyright structure of the project highly depends on the specific circumstances as well as on the applicable law. Generally, a Free Software project maintained by a community can constitute: (1) a collective work,[206] (2) a work of joint authorship (shortly, a joint work),[207] or (3) a derivative work of other programs.[208] First, if a project is a collective work it most probably means that there is a person, or a group of persons that hold copyrights to the project as a whole.[209] This copyright is separate from copyrights in contributions included in the whole project. Second, if the project is a work of joint authorship the joint authors hold "shares" in copyrights to the whole.[210] Third, if a project is a derivative work, its copyright holder has

205 Here, we focus on direct contributions to the program (patches, upgrades, updates). However, there are many community participants who do not write code but contribute to the project in many other ways.

206 "Collective work" is the term used in the U.S. Copyright Act to denote a subset of "compilations" (Sec. 101). Berne Convention uses the term "collection" (Art. 2.5). Although the exact meaning of these or similar terms differs under various laws, for the sake of simplicity we use here the term "collective work" to denote all types of works copyrights of which as a whole belong to one person while copyrights to its building blocks (if any) are retained by respective authors.

207 See: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 90 *et seq*.

208 Other possibilities include, for example, a database or an audiovisual work (see, *e.g.*, Droit & Technologies, *La cour d'appel de Paris prend position concernant la qualification juridique du jeu vidéo*, http://www.droit-technologie.org/actuality-1103/la-cour-d-appel-de-paris-prend-position-concernant-la-qualification-ju.html, in French). Here, we do not elaborate on these possibilities.

209 The copyright holder of the collective work is usually its producer (editor). A producer is a person that creatively arranges or directs the arrangement of various contributions into a collective whole. Depending on the applicable copyright law either the contributors retain their rights to contributions (unless they assign them contractually), or the rights are also granted to the producer as a matter of law. For example, under Polish Copyright Act Art. 11 copyrights to a collective work belong as a matter of law to its producer (or editor), while contributors retain rights to their respective contributions (if such contributions are separable). Similarly, a person that creates a copyrightable collection of various items (that can include copyrighted works) becomes copyright holder of the collection. Many communities or firms that maintain GNU/Linux distributions invoke collective works regulations. See, *e.g.*, OpenSuSE, at: http://en.opensuse.org/OpenSUSE_License, but see Ubuntu, at: http://www.ubuntu.com/community/ubuntustory/licensing.

210 Various laws differently distinguish between a collective work and a joint work. For example, under the U.S. Copyright Act, contributions in a joint work have to be inseparable, while the Polish Copyright Act allows a joint author to exercise the rights in an independent part of the work, thus suggesting that a joint work can constitute of separable parts. Additionally, Polish law provides for a category of "works combined by authors for joint distribution" (Polish Copyright Act Art. 10), which is regulated similarly to a joint work.

to follow directions of the copyright holder of the original in order to exercise the freedoms in the derivative work.

Remarkably, all these three variants can intermingle in practice.[211] For example, we should not rule out a possibility that a particular Free Software project can constitute a collective work created jointly by a group of persons. Namely, not only numerous participants contribute by separate contributions to the project, but also there is a number of persons who organize the contributions or select them to be included in the whole. Additionally, while including the contributions in the whole, some participants may modify them, and in particular create derivative works. For example, in a GNU/Linux distribution, a packager of one program (*e.g.*, a library) selected for inclusion in the distribution might create a derivative work of it, cooperate in the integration of the derivative work in another program with its developer (a program that uses the library), and finally the project maintainers might include both programs in the whole project.

Community participants not only contribute to the community project, but also organize themselves within the community. Depending on the size and nature of the project, the number of contributors, its development stage, *etc*., such an organization takes various forms. In small projects there is usually one person that supervises the maintenance of the whole project, while other contributors restrict themselves to submitting contributions only. In bigger communities the authority and responsibility for the project is distributed. The social structure of an average software community gathered around a medium or large Free Software project usually consists of (1) a few leaders who decide about the development of the project and (2) a number of contributors that follow the leaders. We shall refer to the leaders as *project owners*.

According to Eric S. Raymond (2000), "[t]he owner of a software project is the person who has the exclusive right, recognized by the community at large, to distribute modified versions."[212] More precisely, project owners serve as a source of *official versions* of the project. Unofficial versions are usually understood as versions maintained individually by users outside of the

211 Actually, this seems to be a case of many GNU/Linux distributions such as Debian.

212 Eric S. Raymond, *Homesteading the Noosphere*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/. Project ownership understood as a kind of exclusivity over the maintenance of the project may seem a paradox, as above we extensively explained that under the Free Software licenses every user is allowed to maintain Free Software on their own. In fact, there is no paradox, as long as we read Raymond carefully and note the distinction between (1) the legal right, which is indeed available to everyone as provided for in the licenses, and (2) the right "recognized by the community", which is granted to certain individuals only. Indeed, *project ownership* may not be mistaken for any kind of legal title in software, in particular the copyright to the project, although it may accompany such a title. See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 25 *et seq*.

community.[213] Actually, even inside the communities it is every user's right to modify Free Software for their own use, and to distribute such modifications in some "closed user or development group".[214] In fact, no project owner can legally prevent any user (including community participants) from releasing competing versions of the project on their own. Also, project owners may not legally bind other community members to produce a particular contribution.[215] Straightforwardly speaking, project owners merely accept or reject contributions that flow to the project, and a rejection does not prevent from including the contribution in an unofficial version.[216] In order to exercise this limited authority, project owners gather and manage resources necessary to control the quality of the contributions (both technical and legal), and to integrate the contributions in the official version. The project owners assign these activities between themselves and recruit additional project owners from the community if there is such a need.[217]

213 See, *e.g.*, Alan Cox, *Cathedrals, Bazaars and the Town Council*, at: http://slashdot.org/features/98/10/13/1423253.shtml. (described by the author himself as "a guide to how to completely screw up a free software project"; since the decision to filter out "the masses" did not allow to take full advantage of the bazaar model). Even in such a situation, freedoms are not taken away from users; they can still exercise them and attempt maintain the project individually.

214 Eric S. Raymond, *Homesteading the Noosphere*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/.

215 However, a project owner might be able to legally prevent others from using the project's resources, *e.g.*, a trademark. Actually, trademarks are often used in software communities to indicate that the software originates from the official version of the project, or that it conforms to certain criteria. See, *e.g.*, http://www.linuxmark.org/. Some Free Software licenses, such as the Apache licenses contain prohibitions against using certain markings or otherwise suggesting endorsement by project owners (see: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 17 and 23).

216 Heather Meeker calls it "a free market with a specialist's desk" (Heather J. Meeker, The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008), 25.

217 There are even projects, such as Debian, that have developed quite formal mentoring rules. In Debian, participants from the lower levels of the hierarchy are trained and guided, and there are even recruitment and evaluation procedures in place. See, *e.g.*, Andreas Barth, Adam Di Carlo, Raphaël Hertzog, Christian Schwarz, Ian Jackson, *Debian Developer's Reference*, at: http://debian.org/doc/packaging-manuals/developers-reference/. or Debian, *Debian New Maintainers*, at: http://debian.org/devel/join/newmaint. See also E. Gabriella Coleman, The Social Construction of Freedom in Free and Open Source Software: Hackers, Ethics, and the Liberal Tradition, (Dissertation, University of Chicago, 2005), at: http://www.healthhacker.org/biella/freesoftware.html (Describing the process of adopting a new member of the Debian developer community, consisting of a thorough peer examination of the candidate's technical skills, ethics and the knowledge of legal issues). One could argue that such rules are no longer mere norms, but they are regulated by law and subject to legal sanctions. Indeed, the law usually allows for informal arrangements to constitute source of legal rules. However, we are not aware of any communities that would heavily invoke the law when disputing rights and obligations of project owners *vis-à-vis* community participants.

There is no general rule how project owners organize themselves.[218] There can be no formal organization, such as the "inner circle" in the LINUX kernel community.[219] Other project owners, such as the owners of APACHE, follow quite formal documents that differentiate roles in the community.[220] Many projects formalize themselves even more by using *umbrella organizations* such as foundations or corporations.[221] Umbrella organizations are means for an institutionalized control of project owners over the maintenance of the project.[222] Additionally, it is often the case that such an organization is the holder of valuable project resources, be it servers, development tools, web pages, mailing lists, or copyrights, trademarks, *etc*.[223] The function of any such organization, whether formally incorporated, or being an ad-hoc committee, is to aid project owners in performing various project-maintenance activities.

We shall now proceed to analyse how the maintenance of Free Software projects by software communities affects the limitations and restrictions in the individual exercise of the freedoms. We will address the limitations and restrictions in the already introduced order: license proliferation and incompatibilities (Subsection 4.2.1), license revocability (Subsection 4.2.2), *inter partes* nature of licenses (Subsection 4.2.3), software-related patents (Subsection 4.2.4), contracts with distributors (Subsection 4.2.5), liability rules (Subsection 4.2.6), non-legal regulators (Subsection 4.2.7), closed standards 4.2.8), and regulatory environment (Subsection 4.2.9). Afterwards, we will present our conclusion on the freedoms in software communities (Subsection 4.2.10).

4.2.1 License proliferation and incompatibilities

License proliferation and incompatibilities are differently affected in projects that are collective works, joint works, or derivative works.

---

218 See, *e.g*, Chris Jensen and Walt Scacchi, *Role Migration and Advancement Process in OSSD Projects: A Comparative Case Study*, at: http://opensource.mit.edu/papers/Jensen-Scacchi-ICSE-2007.pdf.

219 See: Gary Murphy, *The Linux Kernel, Blueprints for World Domination*, (table of contents available at: http://kernelbook.sourceforge.net/pkbook.html).

220 See: Apache, at: http://www. Apache .org/foundation/how-it-works.html.

221 Actually, many projects have been founded by corporations (firms) that continue to be their major contributions and community participants.

222 Such organizations are often legally mandated to maintain the project without revoking user freedoms, in the organization's by-laws, articles of association, or even in mere "public deeds" published on the project's web page.

223 Some projects require from contributors to vest some of their rights in the umbrella organization. For example, contributors to OPENOFFICE are required to sign an assignment of joint copyrights to their contributions to Sun Microsystems, Inc., which founded the project on the basis of the STAROFFICE code base (OpenOffice, at: http://contributing.openoffice.org/programming.html#sca.pdf). Similar approach is adopted in other major projects, such as the KDE or MYSQL.

In order to include a contribution in a collective work, an authorization of its copyright holder is necessary. We assume that all Free Software licenses contain such an authorization.[224] More importantly, it is believed that the inclusion of programs under FSD-compliant licenses in a collective work does not create incompatibility problems in the sense that all obligations can be followed separately with regard to each contribution, even though they could not be observed simultaneously with regard to a single piece of software.[225] So, most issues of license proliferation and incompatibilities should not arise in case of projects that constitute collective works, as long as only Free Software is included in such projects.[226] This, however, still implies the need of at least a basic legal audit of software included in the project and its licenses.

In order to create a joint work, the authors have to cooperate in combining their respective contributions. The cooperation necessarily implies that authors agree on such issues as the project's license, not that they merely combine together contributions already subject to a license. Otherwise, the project would rather be a collective work or a derivative work, not a joint work. So, generally there should be no issues related to license proliferation and incompatibilities in projects that constitute joint works.

In order to create or use[227] a derivative work, its author has to obtain

224 Some Free Software licenses express this authorization better than others. Some licenses contain only implied authorization, included in the general consent for the use of the program in any way. Some national laws could require a particular wording for a consent to include the program in a collective work. There is no such specific requirement in the Polish law, but according to the general rules the consent should clearly follow from the license.

225 It is generally believed that the GPL does not require that the whole collective work is licensed under the GPL (partially because of its "mere aggregation" clause) and that it only requires not to restrict user freedoms in the covered work as included in the collective work, not in the whole collective work. However, the inclusion in the collective work might involve creation of a derivative work of the contribution, which may lead to incompatibility problems. See: Free Software Foundation, *GNU General Public License Frequently Asked Questions*, at: http://www.fsf.org/licensing/licenses/gpl-faq.html#GPLInProprietarySystem. See also: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004) 82 and 159.

226 Many communities attempt to maintain projects that include software released under non-FSD compliant licenses. Usually, they keep such software in separate repositories, not distributed together with the rest of the project. Many firms that develop and distribute projects based on Free Software include proprietary programs in the distributions, on the basis of separate agreements with their copyright holders. It is accepted in the market, that such collective works do not infringe Free Software licenses of programs included therein.

227 Various laws differently regulate this issue. Under Polish law the general rule is that the creation of derivative works is not restricted, only the use of them outside of the Polish equivalent of *fair use* requires authorization of the copyright holder of the original. However, under the specific regulation of programs in the Polish Copyright Act (in-line with the Software Directive), any modification of a program is a breach of the author's monopoly if not authorized by the copyright holder. Similarly, under U.S. law, authorization is necessary for the creation of a derivative work.

authorization of the copyright holder of the original.[228] But apart from the authorization, the copyright holder of the original does not have to interfere. Actually, further involvement of the copyright holder of the original would most probably lead to creation of a joint work. In the Free Software scene, derivative works can be developed, distributed, and used even without knowledge of the copyright holder of the original. The author of the derivative work simply exercises the authorization granted in advance in a Free Software license of the original. But the authorization may come bundled with an obligation incompatible with obligations related to other contributions. So, the author of the project has to perform the two-step legal analysis that we described in Section 4.1 when discussing the impact of license proliferation and incompatibilities on the individual exercise of the freedoms.

It follows that negative consequences of license proliferation and incompatibilities on user freedoms are materially minimized in case of projects that constitute collective works or joint works. In case of collective works, even if contributions are licensed under many different and incompatible licenses, their inclusion in a collective work is still possible (provided that they are all licensed under an FSD-compatible license). In case of a joint work license proliferation and incompatibilities issue should not arise at all. Conversely, in case of projects that constitute derivative works of other programs, license proliferation and incompatibilities remain an important limitation of the freedoms. However, as we already noted Free Software projects cannot be easily attributed to one category of works or they can meet the criteria of a collective work, a joint work, and a derivative work at the same time. Precisely speaking, different criteria are usually met at different levels of the project's structure. So, many communities, especially communities that maintain complex projects consisting of numerous programs, have to take license proliferation and incompatibilities into account while maintaining their projects.

Many communities do perform the legal audit necessary in order to overcome license proliferation and incompatibilities. For that purpose, they use the authority of project owners to accept or reject contributions to the official version. For example, many project owners require that the contributors document that they cleared copyrights to the contributions and that they are free to license them under a Free Software license.[229] Usually a specific

228 There is a (criticised) theory under Polish law that the rights to any modifications of a computer program belong as a matter of law to the copyright holder of the original. See: Małgorzata Byrska, *Prawne aspekty modyfikowania programu komputerowego [Legal aspects of modifying computer program]*, 4 Kwartalnik Prawa Prywatnego [Private Law Quarterly], 693, 715 (1996). For a discussion on the status of derivative works in the U.S. law see, *e.g.*, Mark A. Lemley, *The Economics of Improvement in Intellectual Property Law*, 75 Texas Law Review 989, 1022 (1997).

229 The evaluation can be done by humans, but also by intelligent agents, such as the Virtual Stallman, a computer program whose task is to identify all non-free packages in a given GNU/Linux Debian distribution (Virtual Stallman, at: http://alioth.debian.org/projects/vrms/).

Towards_an_Improved_def.indd 136 23-02-2010 10:49:28

FSD-compliant license is required for contributions (*i.e.*, the project's license). Additionally, in many projects elaborate and formal policies have been established for scrutinizing whether a particular contribution conforms with the projects' licensing and other legal requirements.[230] In such projects, licenses of contributions are checked against such policies and any doubts are extensively discussed in attempts to resolve them. Customarily, the record of such discussions is publicly available.[231]

Obviously, such a legal audit involves or even expands both steps of the legal analysis that we found necessary in order to be able to maintain combinations of Free Software programs legally, despite license proliferation and incompatibilities.[232] It allows to identify, which combinations of Free Software programs can be maintained in spite of the proliferation and incompatibilities. Properly organized communities are able to perform such an analysis more effectively than many of the participants individually (and definitely more effectively than an average user could).

In any case the legal audit as such does not lead to a decrease of the number of licenses or to a removal of incompatibilities between them. Nevertheless, if the communities gathered around significant projects implement and follow a well designed and sufficiently strict licensing policy they may indirectly cause such an effect. Namely, contributors who wish to have their contributions included in such projects would be stimulated to adjust their licenses to the policies of the communities. This may make them switch to more popular, compatible licenses.[233] As a result, the communities indirectly support the tendencies that lead to a decrease of the number of licenses and an increase of their compatibility, which we already identified in Chapter 3.

---

230 See, *e.g.*, *Debian Free Software Guidelines* (http://www.debian.org/social_contract) or *Fedora Packaging Guidelines* (http://fedoraproject.org/wiki/Packaging/Guidelines). Actually, many hackers who declare themselves as non-lawyers (by typing a customary "IANAL" abbreviation in almost every communication) often seek compliance with the licenses and the copyright law more diligently than a qualified lawyer would (see, *e.g.*, *The Fedora Extras license audit*, at: http://lwn.net/Articles/218977/; Ubuntu Daily, *Proprietary drivers in Feisty: not by default but easy to activate*, at: http://ubuntudaily.com/2007/03/06/proprietary-drivers-in-feisty-not-by-default-but-easy-to-activate/). Community participants who follow such guidelines quickly identify and remove "non-free" contributions, or at least clearly mark them for a user to be aware of FSD-incompliant code. Sometimes, such contributions are moved to a separate, external repository, such as, *Livna* (http://rpm.livna.org/rlowiki/FAQ).

231 See, *e.g.*, debian-legal mailing list archives, at: http://lists.debian.org/debian-legal/.

232 Given internal project policies, we can distinguish additional level of incompatibility, apart from license incompatibility. Namely, licenses of contributions can be found incompatible with project policies. Such incompatibility is often discussed *e.g.*, within Debian.

233 At the time of this writing it is already the case that licensors rarely draft their own licenses. In the past, however, many licensors did not find already existing model licenses satisfactory. See, *e.g.*, Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004), 85 *et. seq.*, 174. Nowadays, one of the OSI-certified (and accepted by the FSF) model licenses is usually adopted by new projects. Communities augment this standardization trend (*id*. at 175).

4.2.2 License revocability

License revocability is differently affected in projects that are collective works, joint works, or derivative works.

In case of projects that constitute collective works, the applicable law might allow the copyright holder of the whole project to revoke the license to the whole unilaterally. As a result, although particular contributions might remain under their respective Free Software licenses, this would still effectively restrict the freedoms. It would definitely be the case if separate contributions were not as usable as the whole project. So, if the community project is a collective work, license revocability remains a potential threat to user freedoms (to the extent a user would like to exercise the freedoms in a whole project, not separate contributions).

In case of projects that constitute joint works license revocability depends on how the applicable law regulates the scope of activities that each of the joint authors can exercise without the consent of others.[234] In particular, licensing (and revoking the license) of the whole project in one way or another might require unanimity, majority vote, or even could be possible for each of the authors on their own.

In case of projects that constitute derivative works, the exercise of the freedoms in the derivative work requires an authorization of the copyright holder of the original as a necessary condition. So, if the copyright holder of the original revokes the license that contains such an authorization, the respective licensees are prevented from the exercise of the freedoms, even if all authors of derivative works refrain from revoking their respective licenses. Usually, however, contributions that are derivative works are improvements of the project. As a result of the improvements, the original version of the project is not of much importance. It follows that the copyright holder of the original might not have any practical interest in revoking the license.

Let us analyse a specific example of a project that constitutes a derivative work. Assume an original project is subject to a *copyleft* license, and it is improved in many iterations not only by other contributions, but also by the copyright holder of the original. This may lead to a situation where all of

234 Under Polish law, enforcement of copyright is possible to be performed individually by a joint author. Disposing of each author's share in the joint copyright as well as administering a separable contribution is usually also outside of the reach of other joint authors. Other activities usually require either unanimity, or a majority vote. If no agreement is reached, the dispute can be referred to a court. So, it seems that under Polish law a joint author could not unilaterally revoke the project's license, but the removal of the contribution from the project seems possible. Under the U.S. Copyright law, the opposite seems possible (See: Raymond T. Nimmer, *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7, 49 (arguing that in the case of a joint work, a joint author is separately capable of granting licenses to the work as well as individually enforce copyright; it thus seems that the author could also individually terminate the license). See also FN 129.

Towards_an_Improved_def.indd 138 23-02-2010 10:49:28

them are bound by *copyleft* clauses to each other. If any of them attempted to revoke their licenses and distributed their improvements as proprietary, it would most probably lead to a *copyleft* infringement. So, provided that copyrights in a project constitute such a "wickerwork", license revocability may be effectively prevented.

Certainly, in every project, license revocability is subject to practical limitations already presented in Chapter 3. However, given the importance of project owners for the maintenance of projects, it might suffice for the licensor to revoke the license of project owners in order to prevent most users (especially average users) from exercising the freedoms successfully. It follows that license revocability is not effectively prevented by the mere fact that there exists a community that maintains the Free Software project in question. It strongly depends on the actual copyright structure of the project.

However, software communities are able to petrify the license relationship effectively, regardless of such structure. The communities use the authority of project owners for that purpose. Namely, by accepting a particular contribution into a project, project owners present and expose it to the whole community. Such a contribution can still be maintained individually by anyone, but the skills and resources gathered and administered by the community usually outgrow the possibilities of the licensor of the contribution. So, if the license is not revoked, the licensor may continue to benefit from the maintenance provided for by the community. It follows that revoking the license does not make much sense for licensors who cannot maintain their programs individually.

Remarkably, a licensor who does not value the maintenance performed by the community would not take this under consideration when revoking the license. As a result, the respective contribution has to be removed from the project. Substituting a removed contribution usually requires substantial skills and resources, which might not be available for individual users, especially average users. However, a properly organized software community is able to perform such a substitution, so that the maintenance of the whole project can still be provided for.

4.2.3 *Inter partes* nature of licenses

Existence of software communities does not affect the *inter partes* nature of licenses as such. Actually, the fact that there are numerous contributors to a Free Software project may make the enforcement of *copyleft* more troublesome than it already is. For example, if a *copyleft* to a joint work was not observed by a licensee, the applicable copyright law might require that all joint authors stand in a case against the infringing licensee.[235] Even if most of the applicable laws did not regulate these issues in such a way, this would still not affect the *inter partes* nature of the licenses to the benefit of user freedoms.

235 See FN 129.

However, there are numerous examples of Free Software projects to which many participants contribute back despite the fact that software included in the project is not subject to *copyleft*.[236] Actually, even in case of *copylefted* projects, such as the Linux kernel, we would not err in claiming that the most valuable contributions to it were not made as a result of the legal enforcement of *copyleft* clauses.[237] Certainly, one could argue that such contributions are made in performance of obligations other than *copyleft*, since many of them are made by employees of various firms that assign them to work on Free Software projects. Obviously, if such employees contribute to these projects, they perform their obligations towards employers. This, however, does not by itself explain why these firms direct the employees to do so. Naturally, the firms themselves are not obliged to contribute, or they could avoid such an obligation given the *inter partes* nature of licenses.

There have been many attempts to identify the nature of the stimulation that makes numerous and diverse individuals and firms contribute to Free Software projects.[238] Here, it is not necessary to identify what the stimulation really is. It suffices to observe that there are many community participants who find it more valuable to contribute to the project instead of keeping the contributions private. In other words, they value the participation in communities more than trying to maintain the software individually. By accepting their contributions to the official version of the project, project owners enable the community to maintain them, which stimulates the contributors to contribute more effectively than the legal rule of *copyleft*.[239] We would like to refer to this stimulations as *community copyleft*, since it has a result similar to the intended result of *copyleft* clauses.

Clearly, the *community copyleft* does not affect such individuals or firms that find more value in keeping their improvements private. Such "free-riders" can be stimulated with the legal rule of *copyleft* only, with its inherent limitations. However, a properly organized community can serve as a source of improvements, which could substitute the appropriated ones. Definitely, such a community is able to exercise *the right to fork* effectively. Also, in the long run, the "free-riders" would experience severe maintenance prob-

236 Most prominent examples include FreeBSD, OpenBSD, and NetBSD.

237 K.R. Lakhani, B.Wolf, J. Bates, C. DiBona, *The Boston Consulting Group Hacker Survey*, at: http://www.osdn.com/bcg/bcg-0.73/img1.html; Greg R. Vetter, *„Infectious" Open Source Software: Spreading Incentives or Promoting Resistance?*, 36 Rutgers Law Journal 53 (2004).

238 There is a lot of literature trying to explain reasons behind contributing to Free Software projects. See, *e.g.*, Steven Weber, The Success of Open Source (Harvard University Press, 2004); Eric S. Raymond, *The Magic Cauldron*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/; Didier Demazerie, Francois Horn, Nicolas Jullien, *How free software developers work*, at: http://ssrn.com/abstract=1301572 (abstract); Didier Demazerie, Francois Horn, Marc Zune, *The Functioning of a Free Software Community: Entanglement of Three Regulation Modes – Control, Autonomous, and Distributed*, at: http://www.sciencestudies.fi/v20n2DemaziereHornZunePDF; Francesco Rullani, *Dragging developers towards the core*, at: ftp://ftp.unibocconi.it/pub/RePEc/cri/papers/WP190Rullani.pdf.

239 See: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 6, 83, 158, 171.

lems.[240] We assume that in such a case only a few firms would be able to keep the pace of a properly organized community. More importantly, even if they could, such communities would still be able to maintain the project in spite of the "free-riders". Additionally, the reader should be aware that in the overall Free Software community, there are initiatives that support individual licensors and users in enforcing copyleft obligations.[241]

4.2.4 Software-related patents

The existence of software communities does not invalidate software-related patents or make them inapplicable. However, many communities manage to stimulate some patent holders not to use their patents against user freedoms. For example, some major firms involved in the development and distribution of Free Software have publicly declared to refrain from enforcement of their patents in relation to Free Software users in one way or another.[242] The reason for such pledges is similar to the already discussed reasons of not revoking Free Software licenses or contributing to projects irrespectively of *copyleft* limitations. Namely, some patent holders find it more valuable to have their products properly maintained by a community than having to maintain them on their own. Precisely speaking, they care more for a wide adoption of their technologies than for directly benefiting from patent royalties.[243]

---

240 See: Jon Corbet, *How to Participate in the Linux Kernel Community*, at: http://ldn.linuxfoundation.org/book/how-participate-linux-community ("The truth of the matter is that keeping code separate ... is a false economy. ... Incorporation into the mainline solves a large number of distribution and support problems. ... [A]ny out-of-tree code requires constant upkeep if it is to work with new kernels. Maintaining out-of-tree code requires significant amount of work just to keep that code working. ... So code which has been merged into the mainline has significantly lower maintenance costs.")

241 In the U.S. such cases are handled by the Software Freedom Law Center (see: http://softwarefreedom.org), while in the EU they are handled by the GPL-violations project in cooperation with the FSFE (see: http://gpl-violations.org). However, these initiatives do not have their own standing in *copyleft* cases. Rather, they represent some licensors (copyright holders) who may or may not wish to support a particular case. If a user does not convince them to support a particular case, the user has to support the case alone.

242 See, *e.g.*, CNN, *IBM Pledges Free Access to Patents Involved in Implementing 150+ Software Standards*, at: http://money.cnn.com/news/newsfeeds/articles/marketwire/0276035.htm. See also: IBM, at: http://www-03.ibm.com/press/us/en/pressrelease/21846.wss, RedHat, at: http://www.redhat.com/magazine/001nov04/features/patents/, or Novell, at: http://www.novell.com/company/policies/patent/.

243 Nevertheless, software-related patents might allow their holders to restrict user freedoms while at the same time benefit from the maintenance as provided by software communities. Namely, a patent holder could offer a contribution covered by a patent to a Free Software project, under a carefully chosen Free Software license that does not contain any patent license. If project owners accepted that contribution, the patent holder would not enforce the patent against the community, but selectively enforce it, *e.g.*, against competitors who attempted to make commercial use of the project. Assuming that the community were not interested in using the project commercially, this strategy could work for a longer period. See: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 21.

Certainly, there remain numerous patent holders who wish to enforce their patents. If there are many different copyright holders that hold different titles to the project or its parts it might be harder for patent holders to enforce their patents against all of them. However, given the importance of project owners for a properly organized community, it would probably suffice if the patent holders addressed their claims against project owners in order to prevent maintenance of the project effectively. In such a case, given enough resources involved, and a proper legal audit, communities might be able to identify any patents material to the project. Then, if project owners removed the patented technology from the project, or maybe worked around the patents, they might avoid liability and the project could still be maintained. However, it is rarely the case that community participants have expertise necessary in order to search and analyse patents.[244] More importantly, the communities cannot usually afford following any conditions on the exercise of the freedoms that patent holders can impose.

4.2.5 Contracts with distributors

Communities as such do not prevent Free Software distributors from imposing restrictions on user freedoms in relation to programs they distribute. However, if a program is maintained by a properly organized software community, any restrictive contracts with distributors do not make it impossible to obtain the same (or a substitute) program together with the freedoms from the community or from another distributor.[245] Importantly, not only the program as available from the community is free of any restrictions, but it is also properly maintained. As a result, distributors are not able to impose too restrictive contracts in the first place, unless at the same time they can offer users additional benefits that the users cannot obtain from the community (*e.g.*, some additional services on top of the maintenance).

---

244 This usually requires qualified patent attorneys, not hackers. Actually, performing such a research might have negative effects on a community since awareness of a patent might lead to finding of a deliberate infringement, which usually allows patent holders to claim higher damages. This is exactly the reason why many firms explicitly prohibit their developers to read patents. But communities might be more effective in minimizing the risk of patents by identifying *prior art* and thus helping to invalidate patents (see, *e.g.*, Groklaw, *RedHat is Asking for Prior Art*, at: http://www.groklaw.net/article.php?story=20090216150306923; see also: Peer-to-Patent, at: http://www.peertopatent.org/). Also, participants in software communities or simply Free Software enthusiasts can be quite easily motivated to participate in various public events. The most prominent example was the campaign against the so-called "Software Patents Directive". These measures, however, are more a result of the existence of the overall "Free Software Community", not any of the particular software communities. See also: Open Invention Network, at: http://www.openinventionnetwork.com/.

245 A prominent example is CentOS, a GNU/Linux distribution based on the Red Hat's distribution. See: Wikipedia, *CentOS*, at: http://en.wikipedia.org/wiki/Centos.

Also, even if the distributors are able to restrict some users with such contracts, the program remains available without restrictions from the community that can attempt to substitute any additional benefits offered by the distributor.[246] Certainly, all this is possible if only the community that maintains the program is properly organized and possesses sufficient resources, at least equal to the resources of the distributor that attempts to restrict user freedoms.

4.2.6 Liability rules

The fact that there are multiple copyright holders in a Free Software project might affect liability rules to a certain extent. Definitely, under a particular applicable law it might result in shifting the liability for the project as a whole from some participants to others (*e.g.*, from copyright holders of the contributions to the copyright holder of the whole project). Actually, one of the purposes and benefits of organizing the communities under umbrella organizations is that a "corporate veil" is put between copyright holders, developers, and distributors on the one hand, and users on the other hand. This has a more direct legal effect on liability. It supports *the hacker immunity* in removing the liability demotivator from many actors.

But the umbrella organizations usually use liability limitations and warranty waivers and extend *the hacker immunity* on themselves. So, users generally end up in the same situation as they are towards other actors under *the hacker immunity*. However, the organizations can be established for the purpose of delivering well-maintained programs. In such a case community participants would draft their by-laws accordingly. They would also elect representatives whose task would be, in particular, to perform a quality audit of the software. Obviously, the representatives would at the same time be project owners. As a result, the authority of project owners would be intertwined with legally enforceable responsibilities. They could also be given certain legally enforceable rights towards some community participants. All this could contribute towards better quality software being included in the official versions.

It follows that a purpose of a properly organized software community can be to deliver well-maintained programs to users. This is possible if project owners are directed to evaluate the quality of the contributions and if they are directed to require that contributors meet certain quality standards in order to have their contributions included in the official versions.[247] Naturally, projects that are maintained under well-designed quality control rules may still have bugs, lack features, or pose interoperability problems, as is the

246 This constitutes a collective exercise of *the right to fork*.

247 A simplified view on such a quality audit is expressed in the "Linus's law" formulated by Raymond: *given enough eyeballs, all bugs are shallow* (Eric S. Raymond, *The Cathedral and the Bazaar*, at: www.catb.org/~esr/writings/cathedral-bazaar/).

case with any other software. However, a properly organized community can be used to remedy such defects effectively. Also, if such a community exists and it manages to deliver good quality software, then more distributors are willing to offer warranties as well as additional services to users. As a result, the issue of liability for software is minimized, at least as far as the technical defects are concerned. Namely, instead of attempting to claim damages, in many cases users can rely on the community, or on the market for warranties and services, of which the existence is stimulated by the community, for having a defect removed. Owing to a properly organized community, users have also easier access to the market of warranties and services for Free Software.

4.2.7 Non-legal regulators

The existence of software communities does not as such affect the non-legal limitations and restrictions of user freedoms. Definitely, distributors are still able to restrict the freedoms using the architecture, as much as they are allowed to impose restrictive contracts on users. Also, communities do not result in average users having more skills and resources on their own, so that they can effectively exercise their freedoms individually. Additionally, in spite of the *community copyleft* stimulation described above, the mere fact that there are numerous contributors to the project does not by itself result in an increased quality of these contributions, or of the project as a whole. This in turn means that the maintenance of such a project might actually be harder than trying to exercise the freedoms in one Free Software program individually, since all these diverse contributions have to be properly combined and integrated with the project. So, we find that the communities have to deal with additional maintenance issues as compared to the individual exercise of the freedoms.

However, the communities have proven capable of resolving the issues related to non-legal regulators. First, software communities usually constitute a source of Free Software programs without architectural restrictions. Distributors can certainly impose such restrictions on their own, but as long as the community is a substitute source of the program, user freedoms may remain unaffected.[248] This is certainly the case when a community can replicate a service or device that is offered together with the program by a distributor and that constitute the crux of the architectural restriction. But only properly organized communities that hold sufficient resources are able to do so. Still, some services and devices cannot be replicated effectively at all. Second, many communities are able to maintain the programs effectively, including the maintenance for average users. As a result, even though such

248 This depends on actual circumstances. See: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 32 (describing the "embrace and extend" strategy Microsoft used towards the Kerberos protocol).

users do not have the necessary skills and resources, they are able to "outsource" the maintenance to the community. Third, properly organized maintenance involves in particular a quality audit and a procedure for integration of contributions. These lead to properly working programs delivered in the official versions of the project.[249] So, provided the communities organize themselves to perform such an audit and implement these procedures, they are able to overcome transaction costs that an individual user might never be able to cover.

4.2.8 Closed standards

There is no direct relation between the fact that many different participants contribute to a Free Software project and the fact that the project is restricted from using closed standards. Obviously, the bigger the community, the more resources it has to reverse-engineer the standard, to participate in SSOs, or to support an essential facilities case. But the existence of a community as such does not result in such resources being used in order to avoid restrictions on closed standards. The communities do not prevent the rules that lead to closed standards from operation.

However, if a community is a source of a well-maintained Free Software project that interoperates with other programs, designers of standards may have fewer incentives to make closed standards and maintain the respective software on their own, as compared to a situation where there is no such a community. This is a result of the mechanism already described above, the same that stimulates some patent holders not to enforce their patents against the communities or to contribute their improvements to Free Software projects.

Certainly, there remain designers of standards who still find more value in making their standards closed. Provided the community that rejects using such a standard is big and robust enough, it may prevent the standard from becoming popular. If so, the designer might not be able to lock users in a closed standard. But the existence of a community does not guarantee that such a lock-in will not take place. If the users of the project wanted to interoperate with users locked in to a closed standard, their freedoms could be restricted. This means that the communities are not able to remove closed standards from the framework.

4.2.9 Regulatory environment

Above, we often conditioned our findings on what the applicable law exactly provides. Indeed, when numerous participants (from all over the world) contribute to a Free Software project, the issues caused by conflicts of laws

249 See: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly, 2004), 7.

may affect the freedoms even more than when the freedoms are exercised individually. The number and complexity of legal relations increases, in comparison to a situation when a user attempted to exercise the freedoms individually. This means that the communities have to verify the impact of project maintenance in many jurisdictions and the verification might involve more resources than if the freedoms were exercised individually.

It remains to be estimated whether the differences between national laws are of practical importance, such that they could materially affect the operation of the communities. Currently (2009) there are numerous communities that gather participants from all over the world and we are not aware of any issues related to the differences between national laws that hinder their operation materially. This may be because community participants do not have to invoke much legal rules during their collaboration. It seems that much of this collaboration is regulated either by specifically tailored by-laws, or by norms, and there is no need to resort to legal rules. Obviously, such a situation can take place as long as all participants agree. In case of a disagreement, legal rules would be invoked, and the differences between various applicable national laws would be definitely explored. The question remains, whether any disagreement in a community would have to be adjudicated, given *the right to fork* and the fact that any dissidents can establish their own, competing community.

4.2.10 Conclusion on the freedoms in software communities

The communities are able to address many of the limitations and restrictions in the individual exercise of the freedoms. The authority of project owners is instrumental for the operation of the communities as described above. By accepting or rejecting contributions in the official versions of the project, project owners are able to stimulate contributors to prepare the contributions according to various criteria. Contributors are willing to meet these criteria, in particular because of the benefits of community maintenance as compared to the individual maintenance that they would have to perform when contributions were not integrated in the official version.

However, project ownership is not a legal rule and it leads to an indirect stimulation of community participants only. Also, project ownership is not a magical solution. Its exercise has to be organized in order to stimulate contributors in the right direction. It follows that the communities have to be properly organized. Incorporation of umbrella organizations is an important means to this end. But even properly organized communities are not able to remove all the limitations and restrictions of user freedoms effectively. Rather, such communities are only capable of enabling users to exercise their freedoms by materially minimizing the limitations and restrictions. In other words, communities (as much as individual users) can exercise their freedoms as long as they possess sufficient skills and resources. After our analysis we may conclude that properly organized communities are able to gather and manage these skills and resources better than any individual user alone.

4.3 Chapter conclusions

Under the framework, users cannot exercise their freedoms in an undisturbed manner. In particular, they are not able to maintain Free Software individually unless they possess special skills and gather substantial resources. This is definitely the case of an average user, who usually has to rely on others to have the software maintained effectively. Software communities are able to provide for maintenance of Free Software despite many limitations and restrictions of user freedoms. As a result of the inclusion of Free Software programs in community projects users are able to benefit from properly working programs. This does not mean that the maintenance of programs in community projects does not require any skills and resources. Software communities still have to face the same limitations and restrictions that prevent individual users from exercising their freedoms. However, properly organized communities are able to gather and manage the skills and resources necessary to minimize these limitations and restrictions materially. Such communities are able to provide for effective maintenance of Free Software programs.

This means that the threats to the freedoms identified in Chapter 3 still exist in a world of software communities. Only some of them are minimized, but they are not eliminated as a result of the community maintenance of Free Software programs. More importantly, they are materially minimized only if the communities are properly organized.

We are now ready to answer RQ1, which is: "*In what way do software communities affect the current regulatory framework in the protection of user freedoms, as articulated by Stallman?*" We provide the answer in nine concluding statements.

(1) **License proliferation and incompatibilities.** Properly organized communities are able to perform the legal audit necessary to overcome license proliferation and incompatibilities between Free Software licenses. Licensing policies of significant community projects may even indirectly lead (a) to a decrease of the number of the licenses and (b) to an increased compatibility between them, thus supporting the already identified tendencies.

(2) **License revocability.** Some communities create a "wickerwork" relation between all participants, which can effectively prevent license revocability. More importantly, the fact that a program is properly maintained by a community may simply make licensors unwilling to revoke licenses because of the benefits of community maintenance.

(3) ***Inter partes* nature of licenses.** Properly organized communities are able to provide for the *community copyleft* stimulation. It makes some contributors support the project regardless of the legal rule of *copyleft*. This minimizes the limitation that follows from the *inter partes* nature of the licenses.

(4) **Software-related patents.** Some properly organized communities stimulate a number of patent holders not to enforce their patents. However, the communities are not able to prevent such an enforcement completely. In summary, software-related patents remain an important threat in the world of software communities.
(5) **Contracts with distributors.** Properly organized communities prevent distributors to impose too restrictive contracts on users. Alternatively, such communities constitute a substitute source of software available without contractual restrictions.
(6) **Liability rules.** By using umbrella organizations the communities may effectively insulate the actors from liability. At the same time the communities make project owners legally responsible for the quality of software. More importantly, properly organized communities provide for an effective maintenance which leads in particular to a reduction of the number of defects in programs. This minimizes the negative effect that liability rules may have on user freedoms.
(7) **Non-legal regulators.** Properly organized communities perform a quality audit or otherwise control the development of Free Software, so that the software is maintained on an ongoing basis. Such communities may also serve as a source of a given program without architectural restrictions. As a result, communities minimize the negative effect that many non-legal regulators can have on user freedoms.
(8) **Closed standards.** Properly organized communities may contribute to a slowdown in adoption of closed standards. Some communities may even decrease incentives for the designers to make closed standards. However, the communities are not able to remove closed standards from the framework.
(9) **Regulatory environment.** As any other entities, communities operate in a regulatory environment where many national laws have to be taken into consideration. But there are many communities that gather participants from all over the world and there are no signs that any differences between national laws hinder their operation. This means that the exercise of the freedoms in the communities is regulated by customs and trade practices to a material extent. Nevertheless, the regulatory environment remains in the framework despite the communities.

Therefore, we provide the following resolution of PS1, which is "*What are the relations between user freedoms and software communities?*"

Properly organized software communities materially minimize the limitations and restrictions of user freedoms, which in turn allows for an effective exercise of the freedoms in practice. However, even if a properly organized community maintains a given Free Software program, there remain many limitations and restrictions of the freedoms. Particularly, (1) software-related patents, (2) closed standards, and (3) the regulatory environment, remain important limitations and restrictions of the freedoms.

5 User freedoms in eGovernments

We start the analysis of the impact of eGovernments on user freedoms by a telling example of Polish origin. The Polish Social Insurance Office (ZUS) is a government agency responsible for gathering contributions towards future pensions and towards other social insurance benefits.[250] In the late 1990s ZUS began to organize its internal IT system for processing information related to its tasks. The law obliges certain firms to install and use software made available by ZUS in order to submit regular reports about how they fulfil their social insurance obligations. ZUS has made available a program called Płatnik.[251] Since the first release Płatnik has been a proprietary program and has worked under proprietary operating systems only (Microsoft Windows). However, in 2001 a group of hackers led by Sergiusz Pawłowicz started to develop a Free Software substitute for Płatnik. This project, called Janosik,[252] has neither been sponsored by ZUS, nor has it been accepted officially as an equivalent of Płatnik.

Additionally, a major obstacle in the development of Janosik has been the fact that the protocol used by Płatnik to transmit data from users to ZUS was secret (*i.e.*, a closed standard made using the scrambling of interoperability information as discussed in Chapter 4). Without a complete specification of that protocol, it has not been possible for Janosik to operate properly and constitute a real substitute of Płatnik. Naturally, already in 2002 Pawłowicz approached ZUS with a request to reveal the specification of the protocol. The request invoked the right to access public information.[253] ZUS refused to reveal the specification and Pawłowicz appealed first to administrative courts[254] and subsequently to a common court. The common court ordered ZUS to reveal the specification, but ZUS appealed. In April 2007 the 1st instance decision was affirmed by the appellate court.[255] After a slight delay,

250 See: ZUS, at: http://zus.pl.

251 "Płatnik" is a Polish word that stands for "payer".

252 "Janosik" is a name of a legendary Polish (or Slovak) highlander and a robber (Wikipedia, *Juraj Janosik*, at: http://en.wikipedia.org/wiki/Juraj_J%C3%A1no%C5%A1%C3%ADk). He was believed to take from the rich and give to the poor. Naming the project "Janosik" can probably be compared to naming a project "Robin Hood".

253 The right is codified in the Polish Constitution and in the Polish Public Information Act, which is the Polish implementation of the Public Sector Information Directive.

254 NSA decision dated 16 September 2004 (OSK 600/04) a copy on file with the author. At that time ZUS's line of defence was that the protocol contains trade secrets. The court found that when trade secrets are invoked, only the common court can order the publication, so the case was referred to a common court.

255 See: Piotr Waglowski, *Sąd Okręgowy oddalił apelację ZUS w sprawie protokołu KSI-MAIL. Koniec*, at: http://prawo.vagla.pl/node/7222 (in Polish).

in October 2007 ZUS published the specification on its web page.[256] So, after 5 years from the request it is now technically possible to write an alternative implementation of the protocol (*i.e.*, a Free Software substitute to Płatnik).[257]

However, this does not mean that such a substitute can be legally used. Pawłowicz's request concerned the availability of the specification only. He did not ask ZUS to authorize the use of Janosik officially. As a matter of fact, ZUS has not authorized the use of any software other than the proprietary Płatnik, even though it has finally made the specification available, and even though currently there exists explicit legal regulation obliging ZUS not to discriminate users of any software.[258] So, the use of Janosik, or any other Free Software alternative to Płatnik is still legally impossible, or at least legally questionable.[259]

The situation described above is an example of an eGovernment that restricts user freedoms. First, ZUS procured an IT system designed according to a closed standard and able to interoperate with proprietary software only. Second, ZUS has not authorized the use of any software other than the proprietary Płatnik for the communications. Third, ZUS had refused to make the specification available for 5 years. Fourth, to the best of our knowledge ZUS has not offered any support to the Janosik community.[260] As a result, the exercise of the freedoms in communications with ZUS was made impossible and any attempts to change this without ZUS's approval are at least legally questionable, if not straightforwardly illegal. Thus, basing on this particular case we may provide a specific answer the Research Question 2, which was:

RQ 2: *In what way do eGovernments affect the current regulatory framework in the protection of user freedoms, as articulated by Stallman?*

256 The specification is available at: ZUS, at: http://www.zus.pl/bip/default.asp?id=180.

257 As a matter of fact the protocol was reverse engineered long before the case was decided. An example implementation of the protocol made after reverse engineering Płatnik has been made public already in 2003 at: http://www.ibiblio.org/ser/SP-versus-RP/zus/#specyfikacja.

258 Polish Informatization Act, Art. 59.

259 Actually, it is not clear what are the legal consequences of using software unauthorized by ZUS in communications with this agency. Theoretically, since there are procedures for authorization laid down in the Polish Informatization Act, one could argue that the use of unauthorized software is not allowed. We are not aware, however, of any direct sanctions that a user of Janosik could face. *Prima facie*, it also seems that the copyright holders of Płatnik could not claim copyright infringement. However, if ZUS proves that reports were sent without the use of the program authorized by ZUS, ZUS might be able to claim that the obligation to submit the reports was not performed, or performed improperly. This might lead to an indirect liability for the use of Janosik.

260 This is not to say that ZUS was not allowed to perform all these activities. We do not need to decide the legality of ZUS's conduct in this thesis. It suffices to say that the conduct restricted user freedoms. Nevertheless, in the Płatnik case the right to access public sector information provided for in the Polish Public Information Act was breached (the specification of the protocol was held by the court to constitute public sector information). Under the Public Sector Information Directive discrimination of prospective re-users of public information or treating them unfairly is prohibited as well.

In the above context we provide the following specific answer to RQ2. The eGovernment as introduced by ZUS has led to a complete inability of the framework to protect the freedoms of users communicating with ZUS. In particular, users obliged to communicate with ZUS have been directed to use certain proprietary software which means that the freedoms of these users while using that software do not exist at all. These users could theoretically exercise their freedoms outside of the communications with ZUS. However, PŁATNIK requires MICROSOFT WINDOWS, a proprietary operating system to operate.[261] Generally, one computer can work under the control of only one operating system at a time and one operating system allows to operate many other applications, not just one dedicated program. So, using a Free Software operating system would not be practical for many users who were stimulated to use MICROSOFT WINDOWS by the fact that PŁATNIK does not operate on another operating system.[262]

Precisely speaking, the eGovernment as introduced by ZUS affected at least four rules in the framework: (1) *the grant of freedoms*, (2) liability rules, (3) non-legal regulators, and (4) closed standards.

(1) ***The grant of freedoms.*** The eGovernment as introduced by ZUS did not stimulate copyright holders of PŁATNIK to grant users their freedoms; it has remained a proprietary program. Also, the eGovernment has made it legally questionable or at least impractical for users to exercise the freedoms granted to JANOSIK, in communications with ZUS. This effect permeated to the operating system software used by users.
(2) **Liability rules.** Since the use of JANOSIK has not been authorized by ZUS, its use may lead to at least indirect liability.[263]
(3) **Non-legal regulators.** The design of PŁATNIK requires it to be operated on a whole proprietary operating system. This is a restriction of the freedoms using the architecture. Additionally, it has made it more costly for users to exercise the freedoms (*i.e.*, to switch to a Free Software operating system and applications). This stimulated users not to exercise the freedoms as a result of the market regulation.

261 There have been, reportedly successful, attempts to operate PŁATNIK on a GNU/LINUX operating system using WINE (http://winehq.org), but even in such a case PŁATNIK itself remains a proprietary program based on a closed standard.

262 In order to switch between operating systems users would have to reboot their computers, use another computer, or – nowadays – invest in virtualization technologies, none of which is (yet) a practical method. Naturally, it can be argued that most users obliged to use PŁATNIK used MICROSOFT WINDOWS already before. However, this does not have to be the case of all users (at the time PŁATNIK was introduced, DOS was still popular as an operating system to host various accounting software). But, even if this were the case, such users could not abandon proprietary software completely and switch to Free Software after they were obliged to use PŁATNIK.

263 See FN 259.

(4) **Closed standards.** eGovernment as introduced by ZUS has resulted in a lock-in to a closed standard in communications with ZUS. Remarkably, as a result of the 5-year suit only the scrambling of interoperability information was finally prevented, but the protocol has not become an open standard.[264] Since there is still no open standard authorized for communications, users of PŁATNIK continue to be locked in to its closed standard, and to proprietary software.[265]

The above answer allows us to provide a partial answer to the Problem Statement 2, which was:

PS 2: *What are the relations between user freedoms and eGovernments?*

In the above context we provide the following partial answer to the PS2. The eGovernment as introduced by ZUS has restricted user freedoms effectively. No rule in the model of the framework that we reconstructed in Chapter 3 has prevented the restriction. It is particularly noteworthy that even the JANOSIK community has not succeeded in removing restrictions imposed by ZUS completely. We remark that the eGovernment as introduced by ZUS is an example of a Closed eGovernment (*i.e.*, an eGovernment based on a closed standard). So, the following question arise: Is the total restriction of the freedoms a necessary effect of all eGovernments? More precisely speaking, can eGovernments be introduced in a way that does not affect user freedoms, or maybe even in a way that reinforces them? An important part of the answer to this question can be given by answering yet another question. Namely, can user freedoms exist and be sufficiently protected under an Open eGovernment? All these questions are worth answering, they rephrase in some sense RQ2, and they are included in the PS2.

Below we analyse the relation between eGovernments and user freedoms in more detail. We start with a brief analysis of the protection of user freedoms in a theoretical situation where there is no government interference through eGovernment (Section 5.1). Then, we expand our analysis of the relation between user freedoms and Closed eGovernments (Section 5.2). Thereafter, we analyse the relation between user freedoms and Open eGovernments (Section 5.3). Finally, we conclude the chapter by presenting a complete answer to Research Question 2 and a general resolution of the Problem Statement 2 (Section 5.4).

---

264 See Section 2.3 where we presented the definition of open standards for the purpose of the thesis (the EIF v. 1.0 definition). One of the prerequisites for an open standard is that it is maintained by an independent organization in an open decision-making process.

265 Given the fact that the specification has been published, it is now possible to write a Free Software substitute for PŁATNIK. So, the lock-in can be broken provided at least that ZUS authorizes the use of such a substitute and will publish updates to the specification, if any.

5.1 User freedoms in a world without eGovernments

In a theoretical world without eGovernments no-one is required or allowed to use information technologies to communicate with the government. In particular, the government does not use any software for communications. As follows from Chapters 3 and 4, even though Free Software could exist in such a situation, users might be limited or restricted in the exercise of the freedoms. This is definitely the case if they attempted to exercise the freedoms individually. Some or even all limitations and restrictions in the exercise of the freedoms with regard to some Free Software programs or combinations of them could be overcome if properly organized communities maintained projects that included these programs. Nevertheless, even the best software communities are unable to overcome all limitations and restrictions fully.

In Chapter 4 we found in particular that software-related patents, closed standards, and the regulatory environment are the three limitations and restrictions which are the least affected by the communities. Here, we focus on *closed standards*. Certainly, in a world without any government interference some communities are able to stimulate designers of standards to make open standards. Provided that these standards become popular, user freedoms may be unaffected. But communities are not omnipotent and there remain many designers who find more value in making their standards closed than in having implementations of the standards maintained by a community. Then, whenever a user wanted to interoperate with a closed-standards-based program, user freedoms would be restricted. Unrestricted access to interoperable software is possible if only the programs in question used *open standards*.

The adoption of open standards comes with an effort. In Chapter 3 we found that open standards can be obtained using reverse engineering, standard setting, or the essential facilities doctrine. However, after a closer analysis, we concluded that only the standard setting is an efficient mechanism for the adoption of open standards, provided additionally that the procedure of SSOs is properly organized, the resulting open standards are actually implemented in software, and the software becomes popular among users. Whether all these necessary conditions are satisfied depends on various circumstances.[266]

Here, we may conclude that in a world without eGovernments user freedoms are sufficiently protected if only all limitations and restrictions of the freedoms are successfully minimized or even eliminated. This requires in particular the design, implementation, and popularity of open standards, all of which depend on various circumstances. So, one cannot take it for granted that the freedoms are sufficiently protected in every world without eGovernments.

---

266 See: WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.

5.2 User freedoms in Closed eGovernments

The introduction of a Closed eGovernment means that at least some technologies[267] used in eGovernment are based on closed standards. This might be a result of a deliberate choice of closed standards by the government, or an indirect effect of the government procuring software without specifying that it should be designed according to open standards. Obviously, the users of eGovernment services are required to use interoperable software. In economic terms this means that by selecting (either deliberately or inadvertently) a particular standard, the government increases the *network effects* of software designed according to that standard.[268]

There is no apparent reason why designers of closed standards should wish to allow for unrestricted development, distribution, and use of Free Software interoperable with their standards. Actually, we may assume that the mere existence of a closed standard implies that its designer does not want to allow users to exercise the freedoms in software using this standard, but rather wishes to capitalize through restrictions on the use of such software.[269] Otherwise, the designer would make that standard an open standard from the very beginning. Remarkably, the choice of a closed standard by the government and the resulting network effects significantly help the designer to make users use software subject to restrictions, at least when they would engage in government communications.[270] Straightforwardly

267 The term "technologies used in eGovernment" is used here to encompass the software of the public administration as well as of the users of government services.

268 See Subsection 3.2.1, and in particular the discussion about lock-ins and the literature referenced thereto.

269 A rational business strategy would be to capitalize on the government choice of the designer's closed standard as a way for an effective insulation against all competition (both Free Software and proprietary software), or at least for controlling the competition (*e.g.*, by licensing the designer's patents material to the standard to the highest bidder). Other strategies come to mind as well, but they all involve restrictions on the development, distribution, or use of interoperable software. So, they all restrict user freedoms.

270 This may constitute a breach of law by the government. For example, it has been held a violation of the EC Treaty obligations by a member state to grant to an undertaking the power to lay down standards and to check that these standards are met, when the undertaking is itself competing with the companies required to implement the standards (C-18/88, *RTT v. GB-Inno-BM SA*, 11.01.1988, 1991 E.C.R. I-05941). A likewise violation was found in requiring or favoring by a member state the adoption of agreements, decisions or concerted practices or reinforce their effects (66/86, *„Ahmed Saeed Case"*, 11.04.1989, 1989 E.C.R. 00803; See also C-35/96, *Commission v. Italy*, 18.06.1998, 1998 E.C.R. I-03851; C-198/01, *„CIF Case"*, 9.9.2003, 2003 OJ C 264 P. 9). Similarly, member states violate their EC Treaty obligations when they impose a fixed price on certain products, namely the price which has been freely chosen by one manufacturer or importer (13/77, *SA G.B.-Inno-B.M. v. ATAB*, 16.11.1977, 1977 E.C.R. 02115). Also, under the Electronic Markets Competition Directive Art. 2.2, member states have an obligation to ensure that any undertaking is entitled to provide electronic communications services or to establish, extend or provide electronic communications network. Additionally, the provisions of the Standards Directive concerning technical regulations have to be complied with when compliance with a standard is required by the government. The directive allows to use technical regulations in the creation of barriers to trade only to the extent necessary in order to realize public interest.

Towards_an_Improved_def.indd 154 23-02-2010 10:49:31

speaking, it practically guarantees that users will have to accept the restrictions then.[271] Having to accept restrictions on the development, distribution, or use of software is in conflict with user freedoms.

It follows that it is highly likely that after the introduction of a Closed eGovernment the developers of Free Software will not be able to reach sufficient interoperability, or that the distribution or use of such programs will be legally questionable. As a result, Free Software will not match the network effects of proprietary software, and users will be stimulated not to exercise their freedoms, despite any benefits resulting from the freedoms. Only particular proprietary software will properly and legally interoperate using a given closed standard (*i.e.*, only such software that was developed with an authorization of the designer of the standard), and users will be stimulated to turn into a passive audience, which consumes such proprietary software.

We distinguish three types of a Closed eGovernment. First, there is a Closed eGovernment where the use of closed standards in government communications has a spill-over effect on software used by users for other purposes (henceforth "Closed eGovernment (A)"). Second, there is a Closed eGovernment that results in closed standards being used in government communications only, and other software is not affected (henceforth "Closed eGovernment (B)"). Third, there is a Closed eGovernment that attempts to enable users of government services to interoperate with government software using open standards despite the fact that the government software is based on closed standards (henceforth "Semi-Closed eGovernment").

Below, we analyse how each of these three types of a Closed eGovernment affect the framework in the protection of user freedoms. We start with the Closed eGovernment (A) (Subsection 5.2.1). Then, we analyse the Closed eGovernment (B) (Subsection 5.2.2). Thereafter, we analyse the Semi-Closed eGovernment (Subsection 5.2.3). Finally, we summarize and briefly evaluate user freedoms in Closed eGovernments (Subsection 5.2.4).

5.2.1 Closed eGovernment (A)

The fact that users have to use closed-standards-based software in government communications does not as such prevent them from developing, distributing, and using Free Software for other purposes. However, it may make many users unable to exercise their freedoms effectively, even though Free Software remained available for use outside of government communications. This finding is supported by the Płatnik case, described at the beginning of this chapter. In that case certain users were directed to use a whole proprietary operating system (Płatnik required Microsoft Windows).

271 A choice of a closed standard by the government is generally sufficient to conclude that there is such a guarantee. However, the government could aid the designer even more by mandating the use of a particular closed-standards-based program, as was the case with Płatnik.

Theoretically, they could use, and even develop and distribute another operating system, such as a GNU/Linux when not communicating with the government. But since using two operating systems is impractical, for many users the costs of exercising the freedoms outweighed the benefits. As a result, the governmental stimulation had a spill-over effect. Namely, not only the government's choice of closed standards prevented the exercise of the freedoms in government communications, but it also affected them while users attempt to exercise them in communications with non-government parties.

Here, we may conclude that the Closed eGovernment (A) affects *the grant of freedoms*, liability rules, non-legal regulators, and closed standards in a way already described above, when discussing the Płatnik case. Given the spill-over effect, the freedoms are affected both when users communicate with the government and when they use other software, for other purposes. The extent of the spill-over effect may vary depending on actual circumstances.

5.2.2 Closed eGovernment (B)

The spill-over effect that leads to a Closed eGovernment (A) is likely to happen in many situations when the government chooses a closed standard for communications. The support given to the designer of such a standard by the government is often too big to avoid the spill-over, both because of the demand generated by the government itself and also because of the additional demand of users resulting from the network effects caused by the government. Actually, the introduction of a Closed eGovernment (B), which does not involve such a spill-over is possible only if at least the following extraordinary conditions are met together. First, if software used for government communications is highly customized, so it cannot be used for other purposes. Second, if the use of the software does not involve the use of other closed-standards-based software, such as a particular operating system.[272] Third, if only a very limited number of users is directed to use the software. Only in such situations the stimulation which is an effect of a Closed eGovernment would fail to prevent users from using Free Software outside of government communications.

Assuming that a Closed eGovernment (B) is introduced in practice, users are not prevented from exercising the freedoms outside of government communications, but they are also exposed to the already identified limitations and restrictions of the freedoms. In such a case, properly organized software communities might be able to minimize some or all of these limitations and restrictions, with regard to software used outside of government communications.[273] However, it is unlikely that the communities manage to enable

272 Given the increasingly popular shift from delivering software as a product to delivering software services, this condition may have to be rephrased in the nearest future in order to account for the "software as a service" model. See FN 143.

273 See Chapter 4.

users to exercise their freedoms when communicating with the government. For example, Free Software developers gathered in a community such as the JANOSIK community might succeed in reverse engineering the closed standard in question and implement it in a Free Software program. But the use of such a program will still be subject to restrictions related to closed standards (*e.g.*, patents material to the standard[274]), or its use might constitute a breach of the user's obligation towards the government (*e.g.*, if the government did not authorize such a Free Software program[275]).

Here, we may conclude that it is not likely that the Closed eGovernment (B) can be introduced in practice. Using closed standards in eGovernment will most probably result in the spill-over effect that leads to a Closed Government (A). However, assuming that the Closed eGovernment (B) is introduced, it affects the same rules as the Closed eGovernment (A) (*i.e.*, *the grant of freedoms*, liability rules, non-legal regulators, and closed standards), but only to the extent the users engage in government communications. Outside of these communications users experience the normal exposure to the limitations and restrictions included in the framework.

5.2.3 Semi-Closed eGovernment

A Semi-Closed eGovernment attempts to enable users to use open-standards-based software despite the fact that some or all government software is based on closed standards. For example, the back-office software of such an eGovernment could use closed standards but a translation of the communications to an open standard would be provided at the front-office, or at any other point where the interconnection with users takes place.[276] Whether such a *translation facility* would work seamlessly in practice is questionable, since translation between standards usually results in information loss or other problems. But depending on the actual technical and legal circumstances of the closed standard subject to translation, such a facility might allow to develop, distribute, and use Free Software capable to interoperate with the Semi-Closed eGovernment sufficiently. Nevertheless, it should not be expected that such an eGovernment would have the same effect on user freedoms as an Open eGovernment (see Section 5.3 below).

Provided that the translation facility works properly, such an eGovernment would not result in a strong lock-in to closed standards or the spill-over effect discussed above. Not only the users, but also the government

274 To the best of our knowledge there are no patents involved in the PŁATNIK case. Given the publication of the specification made by ZUS as a result of the court order, the trade secret protection should also not apply to the published specification. However, the designer of the protocol retains the control over the standard as a result of the lock in preserved by the government.

275 See FN 259.

276 Another option would be to use an open standard in parallel to any closed standard and to gradually phase out the use of the latter.

might be able to operate the translation facility to switch between closed-standards-based technologies and software based on open standards. Eventually, they may even be able to replace the former with the latter. Depending on actual circumstances, a Semi-Closed eGovernment might minimize the effect of other rules that lead to closed standards as well. For example, if the closed standard in question is restricted using trade secrets or patents, in order to introduce the Semi-Closed eGovernment the government might negotiate a sufficiently broad license that would allow users for unencumbered interoperation with eGovernment using the translation facility. The negotiations would involve indemnity for users of government services against a breach of trade secrets or patents. Additionally, the development of the translation facility by the government (*i.e.*, the contracting for such development) would most probably minimize the effectiveness of the scrambling of interoperability information related to the closed standard in question.

However, the actual impact on the rules that lead to closed standards largely depends on the government's awareness, negotiating power, and the final agreement concluded with the designer of the closed standard.[277] This means that the introduction of a Semi-Closed eGovernment is unlikely unless it is performed diligently. Nevertheless, such an eGovernment might be the only choice (albeit for a transitional period) if the government wishes to transform an already existing Closed eGovernment into an Open eGovernment (we will discuss such a transformation in Chapter 6). In any case we remark that even a properly introduced Semi-Closed eGovernment would not change the closed standard subject to translation into an open one. So, at least some limitations or restrictions of user freedoms will remain in such an eGovernment.

Here, we may conclude that the Semi-Closed eGovernment, if properly introduced, may minimize the restrictions on user freedoms that result from closed standards. It may also avoid affecting *the grant of freedoms*, liability rules, and non-legal regulators to a detriment of user freedoms as other Closed eGovernments do. However, it depends to a large extent on the conditions for the use of the closed standard to provide the translation facility that the government obtains for itself and for the users. Most probably, in order to negotiate proper conditions and maintain the translation facility,

277 Here, we assume that the government does not use its *imperium* or the legislative and judiciary branches in order to dispose the designer of the closed standard of the control over the standard. For example, the government might attempt to grant a compulsory license on such a patent. First, we find such activities not necessarily effective (*e.g.*, under Polish Industrial Property Act a compulsory license can be granted in extraordinary circumstances, which do not include most situations related to eGovernment). Second, when presenting the notion of eGovernment in Section 2.5 we decided to focus on how the government uses its *dominum,* that is how it performs procurement and other contracts-related means. We will discuss how the government may engage *imperium* and the legislative or the judiciary in Chapter 6, while discussing the necessary improvements to the framework.

the government would have to employ significant resources. Even if the government succeeds in introducing a Semi-Closed eGovernment, given the fact that the closed standard subject to translation would still have remained under the control of its designer, the Semi-Closed eGovernment would require an ongoing supervision. Otherwise, user freedoms would still be affected in the long run as it is the case in other Closed eGovernments.

5.2.4 Evaluation of user freedoms in Closed eGovernments

Generally, Closed eGovernments affect the framework in the way already described when discussing the Płatnik case. In particular, Closed eGovernments clearly reinforce rules that lead to closed standards, by *locking* users *in* to the closed standards used in government software. Because of the spill-over effect, Closed eGovernments (A) affect the freedoms with regard to other software as well, even if it is not used in government communications. The impact of Closed eGovernments (B) is limited to government communications only, but whether such an eGovernment can actually be introduced in practice is rather unlikely.

Certainly, Closed eGovernments as such do not make it impossible to introduce open standards, since they do not prevent reverse engineering, open standard setting, or the application of the essential facilities doctrine. However, even if open standards are designed, Closed eGovernments prevent them from becoming popular and this effect usually permeates to government communications as well as to other communications. So, Closed eGovernments definitely minimize efficiency of rules that lead to open standards, in particular the capability of an open standard to become more popular than the closed standard chosen by the government. By doing so, they prevent users from exercising their freedoms.

Users of software affected by a Closed eGovernment cannot exercise their freedoms without limitations or restrictions. Such limitations and restrictions cannot be effectively overcome by any software community, unless the government steps in, for example by introducing a Semi-Closed eGovernment. If this is properly performed, the Semi-Closed eGovernment allows users to interoperate with government software using open standards even though the government software uses closed standards. This allows to develop, distribute, and use Free Software. However, a Semi-Closed eGovernment cannot remove all limitations and restrictions on user freedoms. Moreover, its introduction and sustainability requires due diligence. So, whether it would positively affect user freedoms in the long run is not guaranteed. Nevertheless, a Semi-Closed eGovernment should be taken into consideration when discussing a transformation from a Closed eGovernment to an Open eGovernment.

Below, we identify how each of the affected rules is affected by Closed eGovernments in more detail.

(1) ***The grant of freedoms.*** By choosing (deliberately or inadvertently) a closed standard for eGovernment, the government tips the designer of the closed standard. This results in making the development, distribution, or use of Free Software impractical or legally questionable. As a result, the government either directly stimulates users to use proprietary software, or indirectly stimulates copyright holders not to grant users the freedoms. In either case the government stimulates users not to exercise their freedoms.
(2) **Liability rules.** Given the restrictions involved in the use of closed standards, the development, distribution, or use of Free Software that attempt to interoperate with such a standard may lead to liability.
(3) **Non-legal regulators.** Using a closed standard for eGovernment usually makes it technically impracticable to exercise the freedoms or simply increases the costs of the freedoms. It is a mix of architecture and market regulation. The regulation directly affects software used for government communications, but given the probable spill-over effect, it usually affects other software as well.
(4) **Closed standards.** Using a closed standard for eGovernment helps the designer of such a standard to establish and maintain a lock-in on the government and users. This stimulation may effectively prevent making the standard (or other standards) open even if all other rules that lead to open standards were formally unaffected.

5.3 *User freedoms in Open eGovernments*

The introduction of an Open eGovernment means that all technologies[278] used in government communications use open standards.[279] As is the case with any eGovernment, users have to use software interoperable with eGovernment software. Certainly, the choice of an open standard by the government increases the network effects of software which uses that standard. But interoperability with open standards can be provided by anyone – by Free Software developers and by proprietary software developers. So, no specific software is preferred as a result. In particular, users are not prevented from using Free Software in government communications as well as outside of the

278 See FN 267.
279 For different views on Open eGovernments see: Open Forum Europe, *How Open Can Europe Get*, at: http://www.openforumeurope.org/index.php?option=com_docman&task=doc_download&gid=89&Itemid=102, but see: Initiative for Software Choice, *New EU Public Procurement Directives – Maintaining Neutrality in Software Procurement*, September 2004, at: http://www.softwarechoice.org/download_files/ISC_Legal-Note.pdf. See also: Business Software Alliance, *BSA Statement on Technology Standards*, February 2005, at: http://www.etsi.org/sos_interoperability/Background_papers/BSA_Statement_on_Technology_Standards.pdf.

communications.[280] The choice of open standards as such does not allow anyone to make users accept restrictions on software that may be easily imposed when the government chooses a closed standard.

We distinguish three types of Open eGovernments. First, there is an Open eGovernment that is introduced by using an already existing open standard in government communications (henceforth "Open eGovernment (A)"). Second, there is an Open eGovernment, the introduction of which is possible only after an open standard relevant for government communications is designed (henceforth "Open eGovernment (B)"). Third, there is an Open eGovernment where the government is actively involved in the development, distribution, or use of Free Software, and not just uses software designed according to open standards (henceforth "Supra-Open eGovernment").

Below, we analyse how each of these three Open eGovernments affect the framework in the protection of user freedoms. We start with the Open eGovernment (A) (Subsection 5.3.1). Then we analyse the Open eGovernment (B) (Subsection 5.3.2). Thereafter we analyse the Supra-Open eGovernment (Subsection 5.3.3). Finally, we summarize and briefly evaluate user freedoms in Open eGovernments (Subsection 5.3.4).

5.3.1 Open eGovernment (A)

The introduction of an Open eGovernment (A) comprises the use of an existing open standard in government communications. The government may provide for the use in two ways: (1) by using the architecture and the market regulation, or (2) by using the law. Using the architecture and the market means simply procuring software designed according to an open stan-

280 Here, we assume that the government does not mandate the use of any particular open-standards-based software, but only requires that the communications complies with open standards. However, it may be possible that although the government chooses an open standard, only communications using a certain program is accepted. For example, the Polish Informatization Act provides for software certification procedure which could be used to control which programs are used in government communications in much the same way as project owners control the official versions of Free Software projects. But such a certification by itself does not restrict user freedoms unless a Free Software program that meets a set of objective criteria is refused a certificate.

dard.[281] In such a way, the government contributes towards making the open standard a *de facto* standard. Using the law means that the government mandates the use of an open standard in government communications, and not only procures systems designed accordingly.[282] In such a way, the government contributes towards making the chosen open standard a *de iure* standard.[283] Remarkably, the government can make open standards *de facto* standards using *dominium*. Conversely, making them *de iure* standards requires that the government uses *imperium* (by explicitly obliging certain parties to use open-standards-based software) or even that the legislative branch is involved (in order to provide for a compliance obligation in the law).

Below, we explain how the introduction of an Open eGovernment (A) affects the rules in the framework. For the sake of simplicity, we assign the rules to the following six classes: (1) rules directly related to Free Software licenses, (2) third-party restrictions, (3) liability rules, (4) non-legal regulators, (5) closed standards and open standards, and (6) regulatory environment. After explaining and discussing each of these classes, we present (7) our evaluation of the Open eGovernment (A).

---

281 Government procurement is usually regulated. Under the Public Procurement Directive, which is based on the principles of equal treatment, non-discrimination, proportionality, and transparency, the procurement of technologies based on open standards is not generally prohibited, while procurement of closed-standard-based technologies is questionable. The use of technical specifications (defined in Annex VI to encompass standards as understood in this thesis) in public procurement has to allow „to submit tenders which reflect the diversity of technical solutions" (Recital 29). This relates not only to their impact on the competition, but also on the users and other governmental bodies. The directive obliges to take into account the design for all users whenever possible when formulating specifications (*Id.*). It also requires that the means and technology chosen should be compatible with the technologies used in other member states (Recital 35). Given our definition of open standards they definitely make possible diversity in technical solutions, allow for design for all users, and provide for wide compatibility (through interoperability). Thus, the choice of open standard-compatible technologies in procurement does not infringe these provisions. See also: NOiV (OSSOS), *The acquisition of (open-source) software, A guide for ICT buyers in the public and semi-public sectors*, at: http://ososs.nl/files/acquisition_of_open-source_software_-_text.pdf. Suitability of public procurement for the encouragement of „wider acceptance of open systems interconnection information and data exchange standards through reference to them in purchasing" has been recognized in the EU since at least 1986. See: Decision 87/95 of 22.12.1986 on standardization in the field of information technology and telecommunications (OJ L 36, 7.2.1987, P.31).

282 At the time of this writing (2009) the amendment of the Polish Informatization Act is being processed by the Polish Parliament. The current version of the Act does not explicitly mandate the use of open standards, and the executive order to the Act specifies various standards that can be used in eGovernment, including both open standards and closed standards. Many governments all over the world adopt various instruments, policies, and even laws that mandate the use of open standards or otherwise promote their use.

283 The above understanding of *de facto* and *de iure* standards is not the only one. For example, WIPO understands a *de facto* standard as a standard adopted as a result of the market, while a *de iure* standard as a standard adopted by a SSO See: WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.

(1) Rules directly related to Free Software licenses

Here, we explain how the introduction of an Open eGovernment (A) affects the following rules, which are directly related to the licenses: (a) *the grant of freedoms*, (b) *the right to fork*, (c) *copyleft*, (d) *the hacker immunity*, (e) license proliferation and incompatibilities, (f) license revocability, and (g) *inter partes* nature of the licenses.

The software that is used in an Open eGovernment (A) can be Free Software. But because of the mere fact that open standards are used in government communications *de facto* or *de iure*, users (and the government itself) are not directed to use Free Software, they are only not prevented from using it. As a result users are not prevented from exercising the freedoms, and copyright holders are not prevented from granting the freedoms. But the choice of open standards as such does not stimulate the copyright holders to grant the freedoms to users, or the users to exercise the freedoms.

So, an Open eGovernment (A) does not affect *the grant of freedoms*, as well as other rules directly related to Free Software licenses (*i.e.*, *the right to fork*, *copyleft*, and *the hacker immunity*). It is neutral towards these rules. Consequently, it does not affect limitations and restrictions of the freedoms that are closely related to the licenses, that is license proliferation and incompatibilities, license revocability, and *inter partes* nature of the licenses.

(2) Third-party restrictions

Here, we explain how the introduction of an Open eGovernment (A) affects the following rules, which constitute third-party restrictions: (a) software-related patents and (b) contracts with distributors.

The introduction of an Open eGovernment (A) merely means that the government chooses an existing open standard for government communications. Such a choice, especially if it is made by exercising the *dominium* only, that is by procuring open-standards-based software, does not affect such limitations and restrictions as software-related patents or contracts with distributors.

The government's choice of open standards does not by itself remove the threat of patents related to software. A program that uses a certain patent-free standard may still be found to infringe on patents not material to the standard, but related to the software itself. This, however, depends on the patentability of software- and standards-related inventions in the particular jurisdiction involved.

Also, by choosing an open standard for government communications the government does not materially affect distributors who offer Free Software under contracts that restrict user freedoms. Indeed, no-one is prevented from delivering Free Software for government communications under the terms of their choice (as long as they satisfy public procurement rules or the demand of private users of such software).

So, an Open eGovernment (A) is neutral towards software-related patents and towards contracts with distributors.

Towards_an_Improved_def.indd 163 23-02-2010 10:49:32

(3) Liability rules

Here, we explain how the introduction of an Open eGovernment (A) affects liability rules.

By definition, open standards can be used without risking liability for an infringement of rights related to the standards. So, by choosing open standards for eGovernment, the government makes it possible for anyone to interoperate with government software without risking standards-related liability.

Definitely, an Open eGovernment (A) does not induce users to use closed standards that may be subject to liability. But the liability rules remain in the framework and the development, distribution, and the use of Free Software is still subject to them, for reasons unrelated to standards. So, concluding that an Open eGovernment (A) leads to a relation between such an eGovernment and liability rules would be an exaggeration. The Open eGovernment (A) is neutral towards liability rules.

(4) Non-legal regulators

Here, we explain how the introduction of an Open eGovernment (A) affects non-legal regulators.

The choice of open standards as such does not prevent anyone from restricting user freedoms with the architecture, the market, or the norms. It only affects such restrictions if they are made in relation to closed standards (*e.g.*, the scrambling of interoperability information or the lock-in).

So, an Open eGovernment (A) does not affect most of the existing non-legal limitations and restrictions.

(5) Closed standards and open standards

Here, we explain how the introduction of an Open eGovernment (A) affects (a) closed standards and (b) open standards.

If the government uses an open standard, some parties may find less value in designing their standards as closed, especially if the open standard chosen by the government can also serve other purposes than government communications. This may stimulate designers of standards, in particular SSOs, to make open standards rather than closed standards, *e.g.*, by scrutinizing patents material to standards more diligently in order to satisfy government demand for open standards.

Also, the government's choice of a multi-purpose open standard may make users less willing to use software designed according to other standards. But the developers of such software (1) are not at all prevented to adjust it to the open standard used by the government, and they (2) can continue to design the software according to a closed standard (provided that they obtain an authorization of the designer of the standard and license to any third party rights).

Open standards allow users to choose among many different programs (both Free Software and proprietary software), because any software devel-

oper is able to implement them, and standards-related restrictions cannot be imposed on the software distribution and use. So, an Open eGovernment (A) does not have a spill-over effect similar to a Closed eGovernment (A), it does not favour any particular vendor more than others and it does not lock anyone in to any particular software.

So, an Open eGovernment (A) promotes open standards as compared to closed standards, but it does not remove the closed standards from the framework.

(6) Regulatory environment

Here, we explain how the introduction of an Open eGovernment (A) affects the regulatory environment.

Restrictions and limitations of user freedoms that result from the regulatory environment exist because of differences between national laws. By choosing an open standard for government communications the government cannot affect these differences in any way. Actually, each government has to evaluate a particular standard and software which uses that standard in the light of the applicable law (which does not even have to be its own national law). Some limitations and restrictions that exist under a particular law may not exist under another law, which means that if one government chooses a certain standard then other governments cannot blindly follow without performing at least a rudimentary legal audit.

So, an Open eGovernment (A) does not affect the regulatory environment.

(7) Evaluation of Open eGovernment (A)

Our evaluation is that the Open eGovernment (A) is neutral towards most of the rules in the framework. It materially affects only (1) closed standards and (2) open standards. But it is an indirect relationship, since such an eGovernment does not remove closed standards from the framework. It merely contributes towards the popularity of chosen open standards, which may lead to a decrease in popularity of certain closed standards.

5.3.2 Open eGovernment (B)

In order to introduce an Open eGovernment in a situation where there is no open standard that can be used for government communications, the government has to stimulate the design of such a standard in the first place. One option could be that the government (1) designs the necessary protocol, interface, or a data format (or procures its design), (2) makes sure that there are no restrictions, such as patents material to the standard, and (3) publishes the complete specification with no additional restrictions. But this would not automatically result in the standard becoming an open standard, unless the government had it developed and then maintained in an open decision-

making procedure. So, the government would also have to open the whole procedure to all interested parties.[284]

We do not find the above procedure impossible. We are in particular not aware of any laws that would forbid the government to engage in the making of open standards in such a way, by using its *dominium* only, without having to involve the *imperium* or other government branches. But the purpose of the government as such is not to design or maintain standards. Indeed, SSOs are specifically established for that purpose. So, it seems more reasonable for the government to stimulate the SSOs in the design of the necessary open standard. In particular, the government should diligently participate in SSOs so that they adopt standards of good technical quality. Also, the government should make sure that the SSOs identify and remove (or work around) all restrictions such as patents, as well as that they make complete and comprehensive specifications available without restrictions.[285] This requires extreme diligence, since the fact that a given standard is prepared for the use in eGovernment may increase incentives of other participants to corrupt the procedure so that it leads to making an actually closed standard.[286] Certainly, the government should carefully select in which SSOs to participate in the first place, because not all of them are open to all interested parties and follow appropriate policies, while an open decision-making procedure and such policies are necessary for making open standards.

After the necessary open standard is designed in one way or another, the situation changes into a situation where the government can proceed to introduce an Open eGovernment (A), described in the preceding subsection.

284 Alternatively, the government could stimulate a removal of restrictions on an existing closed standard, a publication of its specifications, and submitting it for review and maintenance to an SSO. The government could use the essential facilities doctrine to directly oblige the designers not to place restrictions on standards that limit their openness. But the doctrine has its limitations, already discussed in Subsection 3.2.2.

285 Government involvement in SSOs is legally possible. For example, the European Commission declares such intervention to the extent necessary to maintain standard-setting organizations as open, transparent, and accountable (Evangelos Vardakas, *The role of government in standards setting: a European View*, 5, in: VADEMECUM ON EUROPEAN STANDARDIZATION, Part II, Chapter 1, at: http://europa.eu.int/comm/enterprise/standards_policy/vademecum/doc/standards_setting_governance_ev.pdf). Also, open standards are actually in-line with basic principles of standardization. The Standards Directive formulates the principle of openness in recital 24, which refers to the organization of standard development procedure, as well as the principles of coherence, transparency, consensus, and independence of special interests.

286 This does not have to involve any corruption in the sense of bribery. It suffices for a participant or a third party to secretly apply for a patent material to a given protocol, interface, or data format before it is adopted as a standard and wait until it is adopted. See: Wikipedia, *Patent ambush*, at: http://en.wikipedia.org/wiki/Patent_ambush. See also: WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.

Here, we may conclude that the Open eGovernment (B) affects user freedoms in the way already described when discussing the Open eGovernment (A). Additionally, such an eGovernment affects closed standards and open standards in a more direct way. Namely, by stimulating the design of open standards the government may affect designers of other standards to participate in the open standard setting procedure instead of attempting to design closed standards on their own. Actually, this may as much induce them to corrupt such a procedure and prevent making the standard open, *e.g.*, by applying for a related patent. The exact outcome, however, much depends (1) on whether and how diligently the government participates in the design of the standard as well as (2) on whether the standard may also be used outside the government communications. Otherwise, such a stimulation may have only a limited effect on other designers.

5.3.3 Supra-Open eGovernment

Government stimulation in an Open eGovernment could go beyond the stimulation already discussed. Assume that apart from using an open standard in government communications, the government stimulates the development, distribution, or use of Free Software in a more direct manner.[287] There are many possibilities for the government to become involved by using *dominium*, without engaging the *imperium* or other government branches (the legislative and the judiciary).

For example, the government might simply procure Free Software.[288] Precisely speaking, the government would either download and use existing Free Software programs, or contract with a distributor for their delivery, possibly together with some additional services.[289] The government could also purchase copyrights to proprietary software and release it as Free Software.[290] Alternatively, the government could procure improvements to an already existing Free Software program and contribute them to the commu-

287 There are many governments that use Free Software already. See: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 39.

288 Procurement of Free Software is possible under the Public Procurement Directive. See FN 281. See also: NOiV (OSSOS), *The acquisition of (open-source) software, A guide for ICT buyers in the public and semi-public sectors*, at: http://ososs.nl/files/acquisition_of_open-source_software_-_text.pdf. Nevertheless, contracting authorities should definitely not rely on the reference to certain labels (See: C 359/93, *Commission v. the Netherlands*, 24.1.1995, ECR 1995, I, 168).

289 Public tendering would most probably not be necessary if the government uses software available publicly and free of charge. See: NOiV (OSSOS), *The acquisition of (open-source) software, A guide for ICT buyers in the public and semi-public sectors*, at: http://ososs.nl/files/acquisition_of_open-source_software_-_text.pdf.

290 The European Union has even designed a whole new model license to be used for releasing government software. See: IDABC, *European Public License*, at: http://europa.eu.int/idabc/en/document/2623/5585#eupl.

nity.[291] The government could also participate in the community in some other way or even become a project owner of projects related to eGovernment themselves.[292]

Given that there are many possibilities for the introduction of a Supra-Open eGovernment in practice, it is hard to determine how it would affect the framework exactly. Below, we discuss some example impacts of a Supra-Open eGovernment on the rules as grouped in the following six classes: (1) rules directly related to Free Software licenses, (2) third-party restrictions, (3) liability rules, (4) non-legal regulators, (5) closed standards and open standards, and (6) regulatory environment. After explaining and discussing each of these classes, we present (7) conclusion on the Supra-Open eGovernment.

(1) Rules directly related to Free Software licenses

Here, we discuss an example impact of the introduction of a Supra-Open eGovernment on the following rules, which are directly related to Free Software licenses: (a) *the grant of freedoms*, (b) *the right to fork*, (c) *copyleft*, and (d) *the hacker immunity*, (e) license proliferation and incompatibilities, (f) license revocability, and (g) *inter partes* nature of the licenses.

The exact impact of a Supra-Open eGovernment on these rules depends on how the government approaches Free Software exactly. Straightforward use of Free Software in eGovernment should not be expected to affect them materially, although the government's demand may stimulate some copyright holders to grant users their freedoms more eagerly. However, if the government actively engages in the development and distribution of Free Software this might have a more direct impact on these rules with regard to the Free Software projects that would be used in eGovernment. Most probably, it will not prevent the operation of the rules that grant or protect user freedoms. Nevertheless, the answer to the question whether it will actually result in any additional protection of user freedoms depends to a large extent on the specific circumstances. Given that there is no single effect that should be expected in every situation, we will return to this issue in Chapter 6,

291 For example, the Polish Ministry of Internal Affairs and Administration has procured a program for editing legal acts and made it available for use by government agencies in order to help them fulfil obligations to prepare electronic versions of various legislation (see: Wojciech Wiewiórowski, *Komunikat dotyczący Edytora Aktów Prawnych [Communication on the Editor of Legal Acts]*, at: http://bip.mswia.gov.pl/portal/bip/21/17881/, in Polish). The program is called EDAP and based on many Free Software components, such as the FCKeditor (http://www.fckeditor.net). EDAP, however, has not been released as Free Software and at the time of this writing it is not clear whether the Ministry would like to animate an EDAP community to maintain it.

292 See: Rishab Aiyer Ghosh, Ruediger Glott, Gregorio Robles, Patrice-Emmanuel Schmitz, Guideline for Public Administrations on Partnering with Free Software Developers (European Commission, Enterprise DG, IDA/GPOSS, 2004), at: http://europa.eu.int/idabc/servlets/Doc?id=19295.

Towards_an_Improved_def.indd 168 23-02-2010 10:49:32

where we propose more specific steps that should be undertaken by governments.

In any case, a Supra-Open eGovernment is capable of indirectly affecting license proliferation and incompatibilities. At the very minimum, when procuring Free Software or contributing to Free Software projects, the government may perform a legal audit of licenses and publish the results of such an audit, thus minimizing the costs of the audit for all users of such software. The government may also stimulate a reduction of the number of licenses and the use of more compatible licenses by contributing to selected Free Software projects. In such a case, the government would support the already identified tendencies that lead to the same effect, and which are supported by many communities as well. However, the government may also cause an increase of the number of licenses, with the European Public License being probably the most important example of such a contribution.[293] How the incompatibilities are affected in such a case depends again on the government's awareness in drafting new licenses.

A Supra Open eGovernment is capable of indirectly affecting license revocability. While procuring rights to software and releasing them under Free Software licenses, the government may guarantee that such licenses will not be revoked (or even straightforwardly donate the software to the public domain[294]). If this is not the case, mere involvement of the government in Free Software projects can serve as an important guarantee that the project will be continued, and stimulate other contributors not to revoke their licenses. But such an effect is not deemed to occur in every Supra-Open eGovernment. Certainly, mere involvement of the government in the development, distribution, or use of Free Software will not make Free Software licenses legally irrevocable.

When such a political choice is made the government may contribute to Free Software projects even if there is no such obligation (*i.e.*, a project is not under a *copyleft* license). This may serve as a stimulus for other contributors to contribute their improvements instead of keeping them private, but it may also cause a free-rider effect by making everyone waiting for the government to contribute, instead of attempting to improve the project themselves. So, government involvement in Free Software projects may have a similar effect

293 See FN 290. Notably, the EUPL includes "compatibility clause" which is intended to minimize the negative consequences of a new model license. Under this clause, the EUPL is expressly compatible with, *e.g.*, GPLv2, but there is no mention about compatibility with GPLv3. So, EUPL and GPLv3 remain incompatible and this will negatively affect user freedoms in case someone attempts to combine software under these licenses.

294 Whether a copyright holder may grant copyrights to the public domain is differently regulated under various laws. Under Polish law its seems not possible for private copyright holders to dedicate their works to public domain, since copyrights attach automatically for a specified period of time. However, Polish Copyright Act does not extend, *inter alia*, to "government documents". Arguably, by procuring copyrights to certain software and declaring it a government document, the government might be able to dedicate it to the public domain.

as the *community copyleft*, but such an effect is not deemed to happen in all Supra-Open eGovernments. Certainly, as such, it will not change the *inter partes* nature of the licenses.

(2) Third-party restrictions

Here, we discuss an example impact of the introduction of a Supra-Open eGovernment on (a) software related patents and (b) contracts with distributors.

The government may contribute to Free Software projects programs or improvements thereof that work around identified patents. Moreover, the fact that a certain Free Software project is used in eGovernment may stimulate patent holders not to enforce their patents, since this alone may lead to an increase of their market in one way or another. But such an effect is not guaranteed, and some patent holders may even find their patents easier to enforce against the government than against individuals or firms. So, there is no direct relation between Supra-Open eGovernments and patents. They remain a threat to user freedoms.

Similarly, a Supra-Open eGovernment does not as such prevent distributors from offering restrictive contracts for the delivery of Free Software or for any related services. In fact, by accepting such a contract the government may even increase the distributor's market power and the ability to impose similar contracts on users. Conversely, when the government starts to offer Free Software programs to users of government services, and regulates the terms of such distribution in a liberal way, this may stimulate other distributors not to use any more restrictive terms. So, whether Supra-Open eGovernments affect contracts with distributors depends on the particular circumstances.

(3) Liability rules

Here, we discuss an example impact of the introduction of a Supra-Open eGovernment on liability rules.

Government involvement in Free Software projects by itself does not affect liability rules. Indeed, the government may regulate the liability for software it procures by requiring additional guarantees, *etc.* Conversely, when the government contributes to a Free Software project it may lead to shifting liability from other participants in the project to the government, or in the sharing of such a liability. However, in any case the government cannot change liability rules that follow from the applicable law, by the mere use, development, or distribution of Free Software.

So, Supra-Open eGovernments do not affect liability rules materially.

(4) Non-legal regulators

Here, we discuss an example impact of the introduction of a Supra-Open eGovernment on non-legal regulators.

The government may require that Free Software projects to be used in eGovernment are free of any or of most of non-legal restrictions. This could

allow users to exercise their freedoms even if other distributors were able to impose non-legal restrictions. Also, the government may cover much of the integration costs necessary for a proper maintenance of a Free Software project, if there is no community that handles the maintenance. As a result, users (especially average users) could benefit from properly working programs even though they themselves lacked the skills and resources necessary to maintain them on their own.

In such a way the government may even substitute or at least significantly aid software communities in the maintenance of Free Software. But whether it affects the freedoms positively or otherwise depends on many circumstances. Obviously, the government is unable to copy all features of software communities, and government involvement may actually lead to a destruction of an existing community. Naturally, the purpose of the government is not to maintain software, while the communities are established specifically for this purpose.

So, whether and how Supra-Open eGovernments affect non-legal regulators depends on how the eGovernment is introduced exactly.

(5) Closed standards and open standards

Here, we mention an example impact of the introduction of a Supra-Open eGovernment on (a) closed standards and (b) open standards.

A Supra-Open eGovernment would incorporate activities related to standards that are undertaken in other types of Open eGovernments. So, it would promote open standards as compared to closed standards, but it would not remove the closed standards from the framework.

(6) Regulatory environment

Here, we mention an example impact of the introduction of a Supra-Open eGovernment on the regulatory environment.

A Supra-Open eGovernment would not affect the regulatory environment for the same reasons as already presented with regard to other eGovernments. It would also face the same limitations and restrictions that are the result of the differences between national laws.

(7) Conclusion on Supra-Open eGovernment

Here, we may conclude that a Supra-Open eGovernment is capable of affecting many rules in the framework. How the rules are going to be affected depends much on how this eGovernment is introduced exactly. We note that the government can contribute to the protection of user freedoms using *dominium*, without resorting to *imperium*, or other government branches, such as the legislative or the judiciary. However, there are limitations and restrictions of user freedoms that every Supra-Open eGovernment is unable to address using the *dominium* only. These are: license revocability, *inter partes* nature of the licenses, software-related patents, liability rules, closed standards, and the regulatory environment.

5.3.4 Evaluation of user freedoms in Open eGovernments

All Open eGovernments prevent the operation of at least some rules that limit or restrict user freedoms, while at the same time they reinforce some rules that protect the freedoms. In particular, by stimulating the design, implementation, and the increase of the popularity of open standards Open eGovernments reinforce rules that lead to open standards and prevent the operation of at least some of the rules that lead to closed standards. If the government does not limit itself to requiring the use of open standards in government communications, but actively engages in the development, distribution, or use of Free Software, it can introduce a Supra-Open eGovernment and affect the framework to an even greater extent.

5.4 Chapter conclusions

In a world without eGovernments user freedoms are subject to the identified limitations and restrictions. Some of them can be overcome by software communities. But whether the freedoms are protected, and in particular whether open standards are designed, implemented, and become popular depends on many circumstances. The government affects these circumstances by introducing an eGovernment.

Generally, Closed eGovernments affect many rules in the framework to a certain detriment of user freedoms. Conversely, Open eGovernments are neutral towards most of the rules, but they affect some of them to a benefit of user freedoms. In order to identify the relations between eGovernments and user freedoms more precisely we distinguished three types of Closed eGovernments: (1) Closed eGovernment (A), (2) Closed eGovernment (B), and (3) Semi-Closed eGovernment. Similarly, we distinguished three types of Open eGovernments: (1) Open eGovernment (A), (2) Open eGovernment (B), and (3) Supra-Open eGovernment.

Each of these six types of eGovernments is in a different relation with user freedoms. Each of them differently affects various rules in the regulatory framework. The differences can be minor (as for example between a Closed eGovernment (A) and Closed eGovernment (B)), but they may also be vast (as for example between a Closed eGovernment (A) and the Supra-Open eGovernment). Nevertheless, we found that neither of the eGovernments (even including the Open eGovernments) affects all limitations and restrictions to user freedoms. Many limitations and restrictions remain out of reach even for the Supra-Open eGovernment. Also, whether some of the limitations and restrictions are affected depends on how such an eGovernment is introduced exactly. Here, we note that in this chapter we focussed on the government acting through the *dominium*. However, in passing we often noted that some of the limitations or restrictions could be addressed only using the *imperium*, or by employing other government branches, such as the legislative or the judiciary.

So, we are now ready to answer RQ2, which is: "*In what way do eGovernments affect the current regulatory framework in the protection of user freedoms, as articulated by Stallman?*" We present the answer in four concluding statements.

(1) **Closed eGovernments.** Closed eGovernments affect the framework negatively from the point of view of user freedoms. Precisely speaking, they affect *the grant of freedoms*, liability rules, non-legal regulators, and closed standards. Given the likelihood of the spill-over effect, these rules are affected with regard to software used in government communications, as well as to other software.
(2) **Semi-Closed eGovernments.** If the government proceeds diligently, it may succeed in the introduction of a Semi-Closed eGovernment. Such an eGovernment enables users to exercise user freedoms despite the fact that closed standards are used in government technologies. But given its limitations it should be taken under consideration only as a transitional eGovernment from a Closed eGovernment to an Open eGovernment.
(3) **Open eGovernments.** Open eGovernments are capable of avoiding the negative effects on the framework that are a result of Closed eGovernment. Still, they do not affect most of the rules, they are merely neutral towards them. Open eGovernments only promote open standards as compared to closed standards, and they do not remove closed standards from the framework.
(4) **Supra-Open eGovernments.** The government is capable of affecting the framework to a greater extent by introducing a Supra-Open eGovernment. However, the exact outcome of such an eGovernment depends much on how it is introduced precisely. Also, even Supra-Open eGovernments are not able to overcome such limitations and restrictions as license revocability, *inter partes* nature of the licenses, software-related patents, liability rules, closed standards, and the regulatory environment. Affecting these limitations and restrictions is possible if only apart from the *dominium*, the government additionally employs the *imperium* or other branches (the legislative and the judiciary).

The above four statements enable us to present the following answer to PS2, which is: "*What are the relations between user freedoms and eGovernments?*" Our answer consists of three statements.

(1) Closed eGovernments are capable of restricting user freedoms effectively, despite the protection provided for in the framework, and despite the positive effect on the freedoms that properly organized communities may have.
(2) Semi-Closed eGovernments enable users to exercise their freedoms despite the fact that closed standards are used by the government.

But their introduction requires much diligence. Still, such eGovernments do not remove the limitations and restrictions that users are exposed to in the world without eGovernments.

(3) Open eGovernments are capable of avoiding the additional restriction of user freedoms that is a result of Closed eGovernments. But they are mostly neutral towards the framework, although they materially affect closed standards. Only Supra-Open eGovernments are able to minimize all existing limitations and restrictions materially. Similarly to Semi- Closed eGovernments, the introduction of Supra-Open eGovernments also requires much diligence. However, any Open eGovernment is still unable to remove all existing limitations and restrictions of user freedoms, unless the imperium or the legislative and the judiciary becomes involved.

6 Proposal of an improved regulatory framework

In this chapter we propose an improved regulatory framework of Free Software. In the construction of the proposal, we use our findings on how the current regulatory framework protects the freedoms in the world of software communities and eGovernments. Below, we recall these findings and then we stipulate the working programme.

We found that (1) access to source codes and (2) access to specifications of standards are necessary conditions of user freedoms. So, we reconstructed a model of the framework, which includes (a) rules for both software and standards, and (b) relations between the rules. In particular, the model includes the rules that follow from Free Software licenses, in which copyright holders of software grant users their freedoms. The licenses provide also for the basic protection of the freedoms. Additionally, the model includes rules that lead to open standards, since only open standards enable users to exercise their freedoms and provide means for interoperability at the same time. Yet, the model includes also many rules which limit or restrict user freedoms either with regard to software, or with regard to standards. These limitations and restrictions are not sufficiently addressed by the rules that follow from Free Software licenses, or by the rules that lead to open standards. So, on the basis of the model, we found that the freedoms are not sufficiently protected under the current framework.

Then, we analysed the operation of the model in a world of software communities. We started the analysis by observing that given the limitations and restrictions existing in the model, users are unable to exercise their freedoms individually in an undisturbed manner. More specifically, users are not able to maintain Free Software individually unless they possess the necessary skills and resources. However, we found that software communities are able to provide for effective maintenance of Free Software programs in projects. After the analysis of the communities we found that for an effective maintenance it is essential that the communities are properly organized. Yet, even such communities do not remove the limitations and restrictions from the framework. They are only able to gather and manage skills and resources necessary to minimize many of them materially. We found in particular that such rules as software-related patents, closed standards, and the regulatory environment continue to affect user freedoms negatively, despite the fact that there exist properly organized software communities. Most of other limitations and restrictions are only indirectly affected by the communities.

Thereafter, we analysed the operation of the model of the framework in a world of eGovernments. We performed a detailed analysis of Closed and

Towards_an_Improved_def.indd 175 23-02-2010 10:49:33

Open eGovernments. We found that eGovernments are capable of materially affecting the circumstances that lead to the design and popularity of open standards. Generally, Closed eGovernments prevent the operation of open standards. They do so by contributing to a lock-in to a closed standard. Conversely, Open eGovernments promote open standards as compared to closed standards, but they are mostly neutral towards other limitations and restrictions of user freedoms. Even Supra-Open eGovernments are not able to overcome all limitations and restrictions to user freedoms. In particular, such rules as license revocability, *inter partes* nature of the licenses, software-related patents, liability rules, closed standards, and the regulatory environment are not affected by them materially.

Our analysis leads us to the conclusion that the current framework does not provide adequate protection of the freedoms in the world of software communities and eGovernments. So, in this chapter we proceed to answer

RQ 3: *How to improve the regulatory framework so that it adequately protects user freedoms, as articulated by Stallman, in the world of software communities and eGovernments?*

In order to answer RQ3 we start by elaborating on the inefficiencies of the current framework and by discussing various possible improvements (Section 6.1). Then, we design the most appropriate improvements in some detail (Section 6.2). Thereafter, we use the designed improvements to construct a proposal of an improved framework (Section 6.3). Finally, we present a summary of the chapter and an answer to RQ3 (Section 6.4).

6.1 Inefficiencies of the current framework and possible improvements

The proposal of an improved regulatory framework of Free Software should address all limitations and restrictions of the freedoms existing in the current framework. Where the current framework, as operating in the world of software communities and eGovernments, promises that it can affect a limitation or a restriction sufficiently without any improvements, the improved framework should not intervene. If this is not the case, adequate improvements should be proposed. Otherwise, the framework will continue to protect the freedoms inadequately, and users will not be able to exercise their freedoms without limitations or restrictions.

In the world of software communities and eGovernments possible improvements can come either from the communities, or from the government. The government can act using only its *dominium*, but it may also employ the *imperium*, as well as the other two government branches (the legislative or the judiciary). So, we distinguish four kinds of possible solutions that can be taken under consideration in the construction of an improved framework: (1) do not propose any improvements, (2) propose improvements that can be undertaken within the communities, (3) propose improve-

ments that can be undertaken by the government using its *dominium* only, and (4) propose improvements that can be undertaken by the government using the *imperium*, the legislative, or the judiciary.

Below, we identify the possible improvements towards each of the identified limitations and restrictions of the freedoms (jointly: inefficiencies). We address all nine of them in the already introduced order. In each subsection we first synthesize the inefficiencies by using our findings from Chapters 3, 4, and 5. Then, we discuss possible improvements to address each of the inefficiencies. We focus on issues specifically related to the purpose of this chapter, so the reader should return to previous chapters for more detail. Thereafter, we present our evaluation of the inefficiencies of the current framework.

6.1.1 License proliferation and incompatibilities

(1) Inefficiencies resulting from license proliferation and incompatibilities
License proliferation and incompatibilities make it necessary for a person who maintains a combination of Free Software programs to perform a legal audit of licenses of all these programs. This is the case even though there are visible tendencies resulting in a decrease of the number of licenses, and in an increase of their compatibility. Only by performing such an audit it is possible to identify all applicable obligations, and all remaining incompatibilities between them. The identified obligations have to be complied with. Programs subject to incompatible obligations have to be removed, or their use in the combination has to be adjusted in order not to trigger the incompatibilities. Following such a procedure is not impossible, but it can be performed only by users who hold the necessary skills and resources. This leads to a decrease of the number of users who become actors on the Free Software scene, *e.g.*, by maintaining Free Software programs. It also diverts at least some of the resources of the active maintainers from the maintenance to the legal audit. In any case, it leads to a decrease of available combinations of Free Software programs.

The evolution of software communities can be perceived as a natural result of many users attempting to cover transaction costs, such as the costs of the proliferation and incompatibilities described above. The communities maintain Free Software programs as projects. However, project maintenance as such does not remove the proliferation and incompatibilities from the framework. Many, if not all, communities still have to deal with the issue. Properly organized communities use licensing policies and the authority of project owners for that purpose. By accepting (to the official version of the project) only the contributions that comply with the project's licensing policies, project owners are able to create a combination of Free Software programs that can be maintained together with all applicable obligations complied with at the same time. Obviously, contributions under incompatible licenses are not accepted, which means that the communities avoid rather than solve the issue. Only the communities gathered around significant proj-

ects can indirectly stimulate copyright holders to switch to more popular licenses, or amend them to provide for more compatibility. Clearly, such communities support the already identified tendencies that lead to the same effect.

Most types of eGovernments do not affect license proliferation and incompatibilities at all. They can be affected by Supra-Open eGovernments only, but how they are affected depends much on how such an eGovernment is introduced. On the one hand, the government may significantly help users from a given jurisdiction in the performance of the legal audit. It may also stimulate a decrease of the number of the licenses and an increase of their compatibility, in much the same way as the prominent communities do. On the other hand, the government may cause the opposite, *e.g.*, if the authorities design new Free Software licenses on their own, without coordinating this activity with already existing community efforts. Also, the actual outcome of a Supra-Open eGovernment on license proliferation and incompatibilities depends on whether the government acts only within its *dominium*, or if it also employs the *imperium*, as well as other branches (the legislative and the judiciary).

(2) Possible improvements to address license proliferation and incompatibilities

As noted earlier, we are currently (2009) in a transitional phase. Free Software licenses have proliferated and as a result many incompatibilities between them have started to restrict user freedoms. But there are tendencies that lead to a decrease of the number of licenses and to an increase of their compatibility. Given these tendencies, we do not find it necessary to propose any extreme improvements of the framework directed at license proliferation and incompatibilities. However, it is crucial that the tendencies are not stopped or reversed. So, we should propose improvements that support the communities in the performance of the legal audit. Namely, the improvements should guarantee that the communities adopt proper licensing policies, and that the policies are followed. The measures could also allow the most prominent communities to stimulate licensors more effectively and to continue to make them switch to more popular and compatible licenses.

From our analysis of license proliferation and incompatibilities we do not derive any guidelines as to which type of eGovernment should be introduced. However, it is still necessary to propose improvements in the framework that would prevent government interference in the identified tendencies. In particular, the government should not hinder the community efforts that lead to a decrease of the number of licenses and to an increase of their compatibility. Additionally, if Supra-Open eGovernments are introduced, the improvements should direct the governments to perform a diligent legal audit of the licenses and to publish results of such an audit, or otherwise serve to the benefit of the users. Governments should also be required to coordinate their involvement in the development, distribution, and use of Free Software (if any) with the communities. Such a coordination is necessary in particular to avoid drafting of more incompatible licenses. It would

also allow for a more effective stimulation of copyright holders to switch to more popular and compatible licenses.

Additional improvements in the framework should be taken under consideration in two situations. First, in case the improvements discussed above should fail to resolve the issue of proliferation and incompatibilities. Second, when the transitional phase reaches its end and it will be possible to indicate but a few model Free Software licenses widely used. In both these situations the legislative branch could attempt to include in the law default terms related to Free Software, or a few sets of such default rules for licensors to choose from. This could be performed in a manner similar to regulating popular contracts in civil codes. In the meantime, the transitional phase of switching between licenses and amending them could be supported by courts. By reviewing the licenses and ruling about the exact scope of rights and obligations of the parties, courts can provide guidelines for Free Software licensors how to amend their licenses, as well as which model licenses are better written than the others. Should a particular license be found invalid in whole or in part, Free Software licensors would promptly amend them or switch to other, better drafted ones.

6.1.2 License revocability

(1) Inefficiencies resulting from license revocability

License revocability allows the licensor to reclaim exclusive control with regard to a Free Software program. The control follows from *the default rule*. After the control is reclaimed, the copyright holder can prevent users from the exercise of their freedoms. License revocability depends on many circumstances, such as the size of the user base, the organization of the distribution of the program, or the applicable law. If these circumstances fail to make it impracticable to revoke the license, the revocation is subject to the discretion of one person only – the licensor. Certainly, it might not be possible to use license revocability in order to prevent all users from using Free Software in private. However, in practice, it would probably suffice that the license is revoked from some major developers and distributors of the program to prevent most users from exercising the freedoms effectively. In particular, it is important that license revocability can prevent effective maintenance of Free Software programs.

Evolution of software communities does not affect license revocability as such. If no additional measures are undertaken, it can be only affected in community projects that consist of multiple iterations of derivative works of an original subject to a *copyleft* license. However, properly organized communities are able to offer licensors benefits that stimulate them not to revoke licenses, even in projects of other types. Namely, they provide for effective maintenance of many contributions under the authority of project owners. In case of licensors that do not find these benefits attractive and choose to revoke their licenses, such communities are still able to stimulate contributions of substitutes by other licensors. Similarly to the effect of the communi-

ties on the license proliferation and incompatibilities, the communities avoid, rather than solve, the issue of license revocability.

As far as eGovernments are concerned, license revocability can be affected by Supra-Open eGovernments only. But this is an indirect relation, and it is not effective towards Free Software programs outside of the reach of the government. Namely, by getting involved in the development, distribution, or use of selected programs, the government may stimulate respective copyright holders not to revoke their licenses. If the government holds relevant copyrights, it may straightforwardly guarantee that the license will not be revoked, or even contribute them to the public domain (if this is possible in the jurisdiction in question). However, the government will not change revocable licenses granted by other licensors into irrevocable, unless apart from the *dominium* it also employs the *imperium*, as well as other government branches (the legislative and the judiciary).

(2) Possible improvements to address license revocability

We are not aware of any license to major Free Software programs having been revoked. Actually, we found that many licensors make pledges to maintain their programs as Free Software in the future. So, we do not find it necessary to propose any improvements that would legally petrify license relations or any other extreme measures. Apart from the above, it has to be borne in mind that licenses are granted by copyright holders, and given the basic principles of the copyright law, they should be free to decide whether to allow the use of their works or not. So, it is not advisable to prevent license revocation for such licensors who are determined to remove their contributions from Free Software and who can do so without infringing rights of other copyright holders. But at the same time it is advisable to allow licensors to make their licenses irrevocable if they find such a need. If under the applicable law a license cannot be made irrevocable by the licensor, such a possibility should be provided for.

Also, we find the stimulation of properly organized communities necessary for the licensors not to revoke their licenses. Precisely speaking, the better a community maintains a program, the less incentives for licensors to control it exclusively. Additionally, the better a community is organized, the easier it can substitute a particular contribution. So, we should propose improvements that prevent stopping or reversing the stimulation of licensors provided for by properly organized communities. Such improvements should guarantee proper organization of the communities.

From our analysis of license revocability we do not derive any guidelines as to which type of eGovernment should be introduced. However, it is still necessary to propose improvements in the framework that would prevent government interference with the community stimulation of licensors. In case of Supra-Open eGovernments we should propose guidelines for governments to support the communities in their stimulation. The guidelines should in particular include the rules for making available software developed or procured by the government.

6.1.3 *Inter partes* nature of licenses

(1) Inefficiencies resulting from the inter partes nature of licenses

The fact that Free Software licenses create *inter partes* relations limits the enforceability of any rights and obligations from a Free Software license by persons other than the parties to the license. In particular, users cannot directly enforce *copyleft* obligations against infringing licensees unless the applicable law gives them standing. In any case, *copyleft* enforcement requires resources of an interested (and eligible) party. This is not impossible, but it leads to a decrease of the number of licensors and users who would claim their freedoms against infringing licensees and succeed. As a result, appropriated Free Software programs cannot be maintained by users, who turn into passive audience, as users of proprietary software do. Determined users can definitely exercise *the right to fork* in such a situation (they can develop a substitute program and release it without restrictions) but this requires skills and resources as well.

Software communities are able to circumvent the *inter partes* nature of the licenses by using the *community copyleft* mechanism. Namely, by providing for effective maintenance of projects consisting of numerous contributions under the authority of project owners, properly organized communities stimulate many parties to release their programs as Free Software regardless of whether the project's license contains a *copyleft* clause. The *community copyleft* is obviously not effective towards such parties who do not find community maintenance attractive. However, properly organized communities may then serve as a source of the substitutes to appropriated improvements. In any case the legal enforcement of *copyleft* clauses remains an option, and the overall Free Software community has developed initiatives that support licensors in enforcing it. Certainly, these initiatives are not able to enforce every *copyleft* infringement and they do not substitute licensors as the main eligible party. Ultimately, if licensors do not support a case, interested users have to support it on their own.

The *inter partes* nature can be indirectly affected by the government, if a Supra-Open eGovernment is introduced. However, the exact result depends much on specific circumstances. Government involvement in the development, distribution, or use of Free Software projects may stimulate some parties to contribute to them too. Nevertheless, it may also cause a free-rider effect, by stimulating many parties to refrain from contributing, if the government takes most or all of the software maintenance burden. In any case, any eGovernment is unable to change the *inter partes* nature of the licenses granted by other licensors as such, unless the government employs the *imperium*, or the legislative and judiciary branches.

(2) Possible improvements to address the inter partes nature of licenses

We find it necessary to propose improvements in the current framework that will make it possible for licensors and users to enforce *copyleft* directly against infringing licensees more easily and effectively. Certainly, these

improvements should make such enforceability legally possible without major limitations, if this is not the case under the applicable law. Additionally, if it is unclear whether under applicable law specific performance (*i.e.*, the performance of *copyleft* obligations) can be ordered in case of *copyleft* infringement, the improved framework should explicitly provide for such a remedy. A question arises, whether it is necessary to change the *inter partes* nature of the licenses further. For example, whether the enforcement of *copyleft* should be left for private parties (licensors and users), or should *copyleft* infringements be prosecuted by the state, as it is usually the case with copyright infringements, which can lead to both civil and criminal liability. State prosecution could be advisable if the licensors and users were unable to counter infringements effectively, even with the help of the initiatives such as the Software Freedom Law Center or the GPL-violations. Definitely, such an extended enforceability should not lead to pushing the freedoms on such users who do not want them at all. But it is advisable to propose improvements that would help users in a situation where licensors do not have the necessary resources to enforce *copyleft*, as well as in a situation where users are unable to obtain the licensor's support of the case.

From our analysis of the *inter partes* nature of the licenses we do not derive any guidelines as to which type of eGovernment should be introduced. We only find it necessary to prevent governments from interfering, to the extent that such an interference would stimulate actors not to contribute to Free Software projects.

6.1.4 Software-related patents

(1) Inefficiencies resulting from software-related patents

A patent may be used to restrict the exercise of some or all of the freedoms, to the extent it covers the use, development, or distribution of a Free Software program. It is not easy to identify a patent infringement *a priori*, mostly because patent information is not readily available in the form that would allow for such an identification. So, no-one can guarantee that a patent is not infringed in the course of the development, distribution, or use of any program, including Free Software. The risk of patent infringement cannot be easily avoided, not even by users who hold substantial skills and resources. Patent-related risk can effectively drive many of the users away from exercising their freedoms. Straightforwardly speaking, it increases the costs of exercising of the freedoms by a factor which is hard to quantify. But these are not only monetary costs of eventual patent royalties or damages. Exercise of the freedoms in Free Software covered by a patent is subject to an authorization of the patent holder. The patent holder can refuse such an authorization or require that certain conditions are met in order to obtain an authorization. Both the refusal and the conditions are in conflict with user freedoms.

Properly organized software communities can stimulate some patent holders not to restrict user freedoms. Some of the holders make even explicit public pledges not to enforce some or all their patents. They see more value

in the fact that projects covered with their patents are properly maintained than in the royalties that could have been collected from users. Still, other patent holders prefer to enforce (or threaten to enforce) their patents, even if the patents cover projects maintained by the communities. The fact that some software communities consist of a large number of participants may make it harder to enforce patents towards members of these communities. But the mere fact that a community is big, is not an adequate protection of user freedoms. It might suffice for the patent holder to enforce a patent towards most important users, such as the project owners, in order to stop the project effectively. However, the communities may be able to gather and manage skills and resources necessary to work around identified patents. As a result, they could stimulate the use, development, and distribution of the project in a way that does not infringe any related patents (*i.e.*, to work around the patents). But they would not be able to use the patented technology, unless they managed to invalidate the patents. Also, they could not avoid liability for eventual past infringements. So, software communities are largely unable to avoid the patent issue, even if they would be able to develop workarounds.

eGovernments do not remove the threat of software-related patents. Even in case of Open eGovernments, software can still be found to infringe a software-related patent, if such a patent was validly granted in a given jurisdiction. Naturally, governments could participate in Free Software projects and support the communities in working around patents. Perhaps the government participation would lead to better workarounds being developed, but this requires a diligently introduced Supra-Open eGovernment. Still, even if such an eGovernment is introduced, it would not allow users to use the patented technology. Also, it would not remove liability for eventual patent infringements from users. In order to do so, the government would have to use the *imperium*, or the legislative and judiciary branches.

(2) Possible improvements to address software-related patents

We find it necessary to propose improvements in the framework that would prevent software-related patents from being granted. This encompasses observing the exceptions from the patentable subject matter, which do not allow to patent computer programs. But there are jurisdictions that do not include such exceptions in their patent law, and we should not expect that all patent laws can be amended to include these exceptions. Also, the exceptions existing in other jurisdictions have often been interpreted not to prohibit software-related patents. So, although we do not abandon to seek improvements that would prevent such patents from being granted, we also find it necessary to propose improvements that would prevent the use of the patents in order to restrict user freedoms, if they are granted.

We envisage a threefold approach for an effective prevention. First, it should be made easier to identify a patent infringement. Second, avoidance of liability for an infringement should be possible at a low cost, while the legitimate interest of the holder of an infringed patent should be observed.

In any case, the exercise of the freedoms cannot be subject to any authorization or condition. Third, the identified community initiatives related to patents should be supported in the improved framework.

As far as eGovernments are concerned, the analysis of software-related patents does not lead us to any specific guidelines as to which type of eGovernment should be introduced. Certainly, the government should avoid the use of technologies patented by persons who use patents to restrict user freedoms. We remark that in the thesis we distinguish between software-related patents and patents material to standards. While the former are not affected by eGovernments materially, the latter are. Namely, patents material to standards are affected by Open eGovernments to a benefit of user freedoms. However, there is a risk that such eGovernments could create additional incentives for patent holders to corrupt the standard setting procedure so that it leads to closed standards. We will search for the necessary improvements related to patents material to standards when discussing improvements that would affect closed standards.

6.1.5 Contracts with distributors

(1) Inefficiencies resulting from contracts with distributors

Contracts are a flexible tool for restricting the freedoms of particular users. Actually, they can be used both to restrict the freedoms as well as to provide for an additional protection of the freedoms. There are practically no limits on how contracts can be used, given the autonomy of the parties principle. In particular, contracts allow to prevent users from maintaining software and to tie them to maintenance services of the contracting distributor. However, contracts affect only such users who accept them, while other users remain free to exercise their freedoms (*ceteris paribus*). Also, many laws already allow to interfere in contracts in order to protect the weaker party or the competition, such as the consumer protection laws and competition laws.

Software communities do not prevent distributors from offering restrictive contracts to users. But properly organized communities can serve as a substitute source of programs without any such restrictions. They are able to compete with many distributors effectively. At the same time they make it possible for other distributors to use programs maintained by the communities in constructing competitive offerings. As a result, the communities indirectly affect the ability of distributors to offer restrictive contracts, because with any such contract a distributor has to offer additional benefits that the communities or other distributors cannot provide at lower price. So, the communities can protect the freedoms against the threat of restrictive contracts with distributors if only the communities are not prevented from becoming competitive sources of Free Software.

eGovernments as such do not make it impossible to offer restrictive contracts to users. This holds true for every type of eGovernments. Even in Open eGovernments software can be subject to restrictions imposed by distribu-

tors. Such restrictions can be minimized by the government only in Supra-Open eGovernments, which are capable of stimulating distributors if introduced diligently. A diligent government is aware of the importance of the freedoms and of the threat of contractual restrictions. Such a government would not accept restrictive contracts related to Free Software procured for eGovernment. But if the government proceeds less diligently and accepts such contracts, it could actually help distributors that restrict user freedoms in gaining market power. The distributors would then use that power by imposing the restrictions on other users. In such a case, an effective protection of user freedoms would require that the government uses the *imperium*, or the legislative and the judiciary branches to regulate the distributors.

(2) Possible improvements to address contracts with distributors

By definition, contracts are concluded between independent parties. It follows that the most natural way of avoiding restrictive contracts is not to accept them. In cases when acceptance cannot be avoided or when negotiation is not possible, competition and consumer protection laws are of particular importance. If a contract related to Free Software constitutes a breach of such laws users should not be prevented from invoking such laws to defend their freedoms. This can be performed under the current framework, so we do not find it necessary to propose any improvements particularly related to the use of competition and consumer protection laws to protect user freedoms. The question whether to provide for any additional measures could be subject to deliberation in case a significant number of users should find their freedoms restricted using contracts, despite the existing laws. For the time being, since we see no such evidence, we should design improvements that would support the communities in sustaining their competitive advantage. Namely, the communities should be supported in their stimulation of distributors, which causes the distributors to offer users material benefits. The improved framework should promote a competition in the Free Software market. For this reason it is advisable to provide for clear guidelines for governments as to how eGovernments should be introduced in order not to help distributors in imposing restrictive contracts on users.

6.1.6 Liability rules

(1) Inefficiencies resulting from liability rules

The hacker immunity attempts to remove all liability from the actors in the Free Software scene. But whether such a waiver is possible, and to what extent it is possible, is always subject to the liability rules of the applicable law. We did not find that the liability rules of any major jurisdiction prohibit such a waiver completely. We rather found that neither shifting all liability towards users, nor forcing it back to the copyright holders, developers, or distributors, is beneficial for user freedoms. The main issue is not who would bear the liability, but whether a user can have any defects in Free Software removed promptly and effectively. If users have access to additional warran-

ties and services related to Free Software, the question of liability and damages is less important. We found that there is a market for such warranties and services, and that liability rules should not materially affect user freedoms if that market continues to operate effectively.

Communities play an important role in the market for additional warranties and services related to Free Software. In particular, properly organized communities provide for a swift removal of many defects identified when the software is already operated by users. As a result, they stimulate distributors to offer additional warranties and services in the first place, and more importantly, they stimulate that the distributors increase quality of their offerings to keep the pace of the communities. Additionally, by using umbrella organizations the communities are able to insulate important actors from liability, while at the same time they provide for legally enforceable responsibility of project owners concerning the quality of software included in the official versions. Also, by maintaining programs properly, the communities make it possible for many parties to offer warranties and services on top of these programs. In such a way, the communities contribute towards a competitive market that can stimulate production of better quality products. So, properly organized communities are able to address the issue of liability rules.

The impact of eGovernments on liability rules depends on which type of an eGovernment is introduced. The government may increase the risk of liability for Free Software by using a closed standard, that is by introducing a Closed eGovernment. The use of Free Software that attempted to interoperate using such a standard could be subject to direct or indirect liability, either towards the government, or towards third parties. Conversely, Open eGovernments are largely neutral towards liability rules. Using an open standard in government communications makes it possible for Free Software to interoperate without risking additional standards-related liability, but liability under other grounds is not affected. Even while introducing Supra-Open eGovernment the government does not affect liability rules directly, although it may stimulate an increase in the quality of software in much the same way as communities do. Nevertheless, if the government proceeds without due care, it may negatively affect the market for warranties and services related to Free Software, even in a Supra-Open eGovernment.

(2) Possible improvements to address liability rules

We find it necessary to propose improvements in the framework that would not allow to prevent the operation of the market for warranties and services related to Free Software. In particular, the improvements should contribute towards a proper organization of software communities so that they continue to be able to remove any defects in the software promptly. These improvements can include government stimulation performed through diligent procurement of Free Software together with the warranties and services. Given the fact that Closed eGovernments affect liability rules to a detriment of user freedoms, we find it necessary to include in the improved framework rules

that prevent the introduction of Closed eGovernments. In case such an eGovernment is already operating, we should propose improvements that allow for a swift transformation from a Closed eGovernment to an Open eGovernment.

6.1.7 Non-legal regulators of software

(1) Inefficiencies resulting from non-legal regulators of software

Non-legal regulators limit the freedoms and they can be used to restrict the freedoms in many ways. Here, we recall the most important examples discussed in this thesis. For instance, the freedoms can be restricted using the architecture. This happens when a Free Software program is used to provide a service or when it is embedded in a device. Additionally, the market causes integration costs (a type of transaction costs) that burden the maintenance of Free Software projects. Generally, both the architecture and the market limit the effectiveness of some of the rules in the framework, such as *copyleft*. In particular, they create barriers that prevent many users to exercise the freedoms individually.

As we already noted, software communities evolve as a result of the attempts of individual users to remove various non-legal restrictions of user freedoms that prevent the individual exercise of the freedoms. In particular, the communities attempt to cover or avoid transaction costs, as well as they attempt to constitute a substitute source of Free Software without restrictions. We found that the communities do not remove the non-legal regulators from the framework. But properly organized communities are able to provide for maintenance of Free Software despite these regulators. For that purpose it is crucial that the communities are organized properly. However, many communities are unable to avoid restrictions, even if they are properly organized. This is in particular the case if the restriction of the freedoms is a result of linking Free Software with services or embedding it in a device.

The impact of eGovernments on non-legal regulators depends on which type of an eGovernment is introduced. A Closed eGovernment usually makes it technically impracticable to exercise the freedoms or simply increases the costs of the freedoms. It is a mix of architecture and market regulation, which often permeates to other software, not just the software used for government communications (the spill-over effect). Conversely, Open eGovernments are to a large extent neutral towards non-legal regulators. This means that they do not prevent non-legal limitations and restrictions from operation. However, a diligent government may introduce a Supra-Open eGovernment in such a way that limits the negative impact of the non-legal regulators on user freedoms. In order to do this, the government has to pay due care to how the eGovernment is introduced exactly.

(2) Possible improvements to address non-legal regulators of software

Due to the existence of non-legal regulators, we find it necessary to propose improvements in the framework directed both (1) at software communities

and (2) at eGovernments. The improvements should support the communities in their efforts to deliver Free Software without restrictions. In particular, the communities should be able to substitute services or devices that are used to restrict user freedoms. This means that the improvements should support the communities in organizing themselves in order to maintain Free Software programs. Given the fact that Closed eGovernments restrict user freedoms by affecting non-legal regulators, we find that it is necessary to include in the improved framework rules that prevent the introduction of such eGovernments. If a Closed eGovernment is already operating, we should propose improvements that allow for a swift transformation from a Closed eGovernment to an Open eGovernment.

6.1.8 Closed standards

(1) Inefficiencies resulting from closed standards

Closed standards restrict user freedoms as far as users attempt to exercise the freedoms and provide means for interoperability at the same time. Closed standards depend on a combination of (1) trade secrets, (2) patents, (3) scrambling of interoperability information, and (4) lock-ins. Developers of Free Software may sometimes succeed in implementing a closed standard in a Free Software program, but the use of such a program is either impractical or legally questionable (if not straightforwardly illegal). The impact of closed standards on user freedoms might be overcome using reverse engineering, if the interoperability information is scrambled or protected using trade secret. But reverse engineering is usually not a practical method, and it may even not be sufficient to reverse-engineer a standard if it is covered by a patent. The fact that patents are legal monopolies creates incentives for patent holders to direct a standard setting into adopting closed standards covered with their patents. Afterwards, they are able to control whether and how users exercise their freedoms when using the standard. As a non-exclusive alternative to patents, designers of standards can use switching costs in order to lock users in to a closed standard and prevent their migration to open standards. Sometimes, it might be possible to open a closed standard using the essential facilities doctrine, but the doctrine has material limitations.

Closed standards lead in particular to restrictions on maintenance of software. They make it illegal or impractical for individual users to maintain Free Software, as far as interoperability is concerned. Communities experience the burden of closed standards as well, but they are also able to minimize it to some extent. Software communities affect closed standards indirectly. Some of them may stimulate designers of standards to make their standards open, instead of seeking ways to capitalize on restrictions. Big and robust communities may even prevent wider adoption of closed standards and contribute to the popularity of open standards. However, it seems more likely that the communities have neutral effect on closed standards. Most communities can at best succeed in enabling users to use open standards in

their projects, but cannot prevent the use of closed standards in other software.

eGovernments are capable of affecting closed standards both to a benefit of user freedoms, as well as to a detriment of user freedoms. If a Closed eGovernment is introduced, the government helps the designer of the given closed standard to establish and maintain a lock-in both on the government and on the users of government services. Given the likely spill-over effect of the Closed eGovernment on other software, this stimulation can prevent open standards from being designed or becoming popular. However, the government might attempt to use a closed standard and not restrict user freedoms at the same time. This can be done through a Semi-Closed eGovernment, which is based on a facility that provides translation from the closed standard to an open standard at the point where interconnection with users takes place. Nevertheless, Semi-Closed eGovernments do not remove all restrictions related to closed standards, and they are not easy to introduce in practice.

Open eGovernments make the restrictions related to closed standards not applicable to users of government services. Such eGovernments do not remove the closed standards from the framework, but under certain circumstances they may prevent such standards from becoming popular. The government may even participate in SSOs and stimulate them not to design closed standards. This, however, requires diligence, since such a participation might create additional incentives to corrupt the procedure and direct it to the design of closed standards. Even Supra-Open eGovernments do not remove closed standards from the framework.

For an adequate protection of user freedoms closed standards have to be substituted with open standards. Open standards allow to exercise the freedoms and provide means for interoperability at the same time. Only properly organized standard setting performed by SSOs can lead to open standards (as summarized above, reverse engineering and the essential facilities doctrine are not sufficiently effective). In order to design an open standard, SSOs have to (1) follow an open decision-making procedure, (2) scrutinize standards in search of any related restrictions, and (3) eliminate the restrictions. It follows that the SSOs should follow properly drafted policies. But even most diligent SSOs are usually unable to enforce their policies against third parties, not to mention problems in enforcing the policies against the participants of an SSO. Unless the SSOs succeed in designing open standards and unless such standards become popular, user freedoms are restricted.

(2) Possible improvements to address closed standards

The current framework contains rules that result in the design of open standards and that increase their popularity. But given the opposite rules that lead to closed standards, we find it necessary to propose improvements that would promote open standards better. Since we found that an open standard setting is the most promising mechanism for the design of open standards the improvements should be mostly directed at SSOs. SSOs that design open

standards should be promoted. The improvements should guarantee that SSOs adopt proper internal regulations (policies) and such regulations should be observed by the participants. This implies exploring measures that make the regulations more effective. In particular, third parties should be prevented from making a closed standard out of a standard designed by an SSO under the proper regulations. Given the fact that the regulations of SSOs usually do not bind third parties, additional improvements should be provided for, which are able to affect third parties.

However, it is not easy to propose improvements capable of reaching the above goals directly, that would be practicable at the same time. First, many entrepreneurs participate in the SSOs precisely because of the expected royalties from patents material to standards. It would probably not suffice to remove such an incentive in order to make SSOs design only open standards. So, any proposed improvements should provide for an alternative incentive, which would still be compatible with open standards. Second, SSOs cannot afford regulating the conduct of the participants too far because this could lead to a cartel, or trigger competition regulations in another way. So, any improvements should not increase this risk, while at the same time effectively bind the participants. These are often mutually exclusive goals. Third, non-participants could be affected if only extreme measures were implemented. Such extreme measures would encompass (1) amending patent laws with exceptions from the patentable-subject matter in relation to standards, (2) extending of the essential facilities doctrine, and (3) prohibiting using trade secrets or scrambling of interoperability information. We are not convinced that even these extreme measures would reach the goal of promoting open standards, not to mention that many of them would require reformulation of whole legal systems.

Although we do not abandon direct measures completely, we find it advisable to explore indirect measures as well. We find that there are many improvements that could be provided for by the government during the introduction of an Open eGovernment, which would not trigger the dilemmas related to direct measures. Namely, when discussing Open eGovernments in Chapter 5 we noted that if the government uses an open standard it may stimulate designers of standards, in particular SSOs, to make open standards rather than closed standards. In other words, government demand can create incentives that could substitute the incentive related to enforcing patents material to standards. This can increase the popularity of open standards especially if the open standard chosen by the government can be used in other types of communications as well. At the same time, given the fact that open standards can be implemented by everyone, such a stimulation should not affect anyone negatively. It follows that we should propose improvements in the framework that lead to introduction of Open eGovernments. If there is a Closed eGovernment already present, the improvements should lead to a swift transformation into an Open eGovernment.

Given the already existing rules that lead to open standards, which are promoted by software communities, our improvements should not prevent

the communities from operation. If possible, the government should properly introduce a Supra-Open eGovernment, in which it would cooperate with the communities in promoting open standards. This means that we should propose regulation of eGovernments that provides for the introduction of such eGovernments. Obviously, the improvements should explicitly prevent the introduction of Closed eGovernments, since such eGovernments directly contribute to lock-ins to closed standards.

6.1.9 Regulatory environment

(1) Inefficiencies resulting from the regulatory environment

Every jurisdiction follows its own law. As a result, user freedoms may vary throughout jurisdictions. More precisely, we can expect different impact on user freedoms of the identified limitations and restrictions under different national laws. As noted earlier, we are not aware of any law that would make Free Software licensing impossible at all, but the laws can result in a different scope of the rights and obligations of various actors and the audience in the software scene. This does not make the exercise of the freedoms impossible, but it increases their costs. In particular, it results in a decrease of the number of prospective maintainers of Free Software that would like to undertake maintenance involving interstate issues. In other words, differences between national laws serve as resistors of the worldwide flow of Free Software.[295]

Software communities operate in such a situation. They often include participants from all over the world. Such a multinational membership may even lead to an increase of the complexity of the legal issues as compared to a situation when users attempt to exercise the freedoms individually. Indeed, the communities as such do not affect the differences between national laws and have to take them into account as any other actor in the software scene operating worldwide. But at the same time the communities have developed customs and trade practices that are often followed by many actors regardless of what the applicable national law has to say.

eGovernments are usually introduced nationwide, not worldwide. So, the governments rarely have to take into account more than one national law. This, however, may be the case if a particular piece of software is subject to another law pursuant to a choice of law clause. But the major impact of the regulatory environment on eGovernments usually means only that a particular government has to adjust its eGovernment strategy to its own national law. It means that an eGovernment scheme designed for a particular jurisdiction might not work properly in another jurisdiction.

---

295 See: Eben Moglen, *Anarchism Triumphant: Free Software and the Death of Copyright*, at: http://emoglen.law.columbia.edu/my_pubs/anarchism.html (proposing "Moglen's Metaphorical Corollary to Faraday's Law": "Wrap the Internet around every brain on the planet and spin the planet. Software flows in the wires. It's an emergent property of human minds to create.")

(2) Possible improvements to address the regulatory environment
We find it necessary to propose improvements that would remove the negative impact that the differences between national laws have on the freedoms. This requires a harmonization of the regulatory environment. If under a particular national law there is a provision that restricts user freedoms, the improved framework should result in such a provision being removed or amended. In this thesis it is not possible to identify all such provisions in every national law that could apply. In the course of our analysis we identified some examples, and we will return to them when selecting improvements in the next section. We also found that exercise of the freedoms is not materially affected by the differences between national laws in properly organized software communities. As already noted, it seems that much of collaboration within communities is regulated either by internal regulations (by-laws) or by customs and trade practices, and there is no need to resort to external legal rules. Even in case of a disagreement between participants there is rarely a need to adjudicate, given *the right to fork* and the fact that dissidents from a community can establish their own, competing community. The differences between these laws will become important if communities fail to regulate their participants using by-laws, customs, and trade practices only.

It follows that apart from proposing improvements that would remove the most extreme restrictions of the freedoms from particular national laws, we should explore improvements that would support the communities in maintaining their projects without resorting to adjudication under national laws too often. At the same time, when a Free-Software-related dispute is brought to court eventually, the court should be required to take into account the custom and trade practices of the communities, so that different national laws could be applied in a harmonized manner in relation to user freedoms.

6.1.10 Evaluation of the inefficiencies of the current framework

Under the current framework, user freedoms are not adequately protected for four reasons. First, there are limitations and restrictions present in the framework itself. Second, these limitations and restrictions are not removed from the framework by software communities, although properly organized communities are able to avoid or at least minimize many of them. Third, some of the limitations and restrictions are reinforced by Closed eGovernments. Fourth, Open eGovernments are not capable of removing all limitations and restrictions, unless the governments employ not only the *dominium*, but also the *imperium* or the legislative and the judiciary branches.

The following limitations and restrictions are the least affected: (1) software-related patents, (2) closed standards, and (3) the regulatory environment. However, the proposal of the improved framework should contain improvements related to all identified inefficiencies. So, we proceed to design appropriate improvements to remedy all identified inefficiencies.

Towards_an_Improved_def.indd 192 23-02-2010 10:49:35

6.2 Appropriate improvements in the framework

In this section we design in detail appropriate improvements that address the inefficiencies discussed in the previous section. We group the improvements into the following six classes: (1) proper organization of communities, (2) regulation of eGovernments, (3) Free Software legislation, (4) restriction of software-related patents, (5) promotion of open standards, and (6) internationalization of the framework.

6.2.1 Proper organization of communities

Proper organization of communities is necessary for the adequate protection of the freedoms. It is in particular necessary to overcome such inefficiencies of the current framework as license proliferation and incompatibilities, license revocability, *inter partes* nature of licenses, contracts with distributors, liability rules, as well as the non-legal regulators. Proper organization of communities encompasses adoption of licensing policies that should be strictly followed when performing the legal audit of all programs included in community projects. Given the fact that many prominent communities have already elaborated such policies and published them, we propose that other communities study these policies and build upon them. As a result, each community should adopt their own tailored policies. The policies would regulate requirements for any contributions accepted to the project, both in terms of acceptable licenses, as well as in terms of the procedure for clearing copyrights required from participants in the community.[296]

All communities operate in a changing legal environment. So, the communities should elaborate a method for amending their policies if there is such a need. Also, the policies should allow the legal audit to be performed in a continuous manner, not as a one-time activity. In case the policies are amended, appropriate procedures should be in place to verify decisions undertaken on the basis of the previous policies. A good quality policy would allow the community at any given moment of time to identify to whom the copyrights to a particular contribution belong, and to confirm whether the community is in compliance with all applicable obligations. In particular, the policy should allow the community to avoid any incompatibilities between licenses.

Communities should also make sure that the policies are followed in practice. Namely, only contributions that meet the specified criteria should be accepted by project owners to the official version. To this end, project owners should follow best community practices such as documenting the legal audit, communicating their decisions to the public, providing reasoning and review of decisions. It should be considered whether to make these

296 See: Software Freedom Law Center, *A Legal Issues Primer for Open Source and Free Software Projects*, at: http://www.softwarefreedom.org/resources/2008/foss-primer.html.

the legal obligations of project owners, for the performance of which they could be responsible and accountable.[297] Definitely, these obligations should be mirrored with certain legal rights of project owners against community participants, if the prestige of being allowed to contribute to the official version is not sufficient to stimulate the participants to follow the lead of project owners. Regulation of rights and obligations of project owners can be provided for by incorporation of umbrella organizations, with sufficiently elaborate by-laws. The by-laws should in particular regulate the election of project owners,[298] their term of office, control over them, and their rights and obligations.[299]

297 Customs and trade practices already existing in the communities should be taken under consideration. If a community is capable of properly organizing itself without resorting to legal rules, legal regulation might not be necessary at all. See: Karl Fogel, *Producing Open Source Software. How to Run a Successful Free Software Project*, at: http://producingoss.com/. See also: Lucie Guibault, Ot van Daalen, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006) 27.

298 By-laws of umbrella organizations should codify already existing methods of acquiring project ownership. Raymond (2000) enumerates three of them: (1) founding a project, (2) "passing the baton", and (3) claiming ownership of a project (Eric S. Raymond, *Homesteading the Noosphere*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/). First, founding a project may be done by writing a program from scratch by one hacker or one firm alone. The most prominent example is probably the Linux kernel. Sometimes, the program is prepared by a group of individuals or firms who should then be jointly considered as project founders. For example, the Apache WWW Server, although the basis for that project was the httpd server prepared initially by an academia and released as public domain later on. Actually, there are even cases when Free Software projects were founded by firms that previously developed them as proprietary software. For example, Netscape (released as Mozilla), OpenOffice. Second, "passing the baton" means transferring of project ownership to an individual or a group appointed by a previous owner. Third, according to Raymond, claiming ownership occurs in situations when the previous owners abandon the project without transferring the ownership to anyone. This may be a result of the owners losing interest in exercising their "right recognized by the community" and abandoning the project. Examples of the second and third method can be often found within Debian, where various packages often change their maintainers. See also: Andrew M. St. Laurent, Understanding Open Source and Free Software Licensing (O'Reilly 2004) 174.

299 According to Raymond (2000), the social status of a particular community participant depends on the value of contributions (See: Eric S. Raymond, *Homesteading the Noosphere*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/). It is often the case that the most valuable contributors have the most to say about the development. By-laws should regulate such a meritocracy, not substitute it. Also, various authors, such as Levy (1985), suggest that there exists a hacker ethic, consisting of norms such as "all information should be free". (Steven Levy, Hackers: Heroes of the Computer Revolution, (New York, Dell, 1985). See also: Eric S. Raymond, *A Brief History of Hackerdom*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-history/index.html; Eric S. Raymond, *How to Become a Hacker*, at: http://www.catb.org/~esr/writings/faqs/hacker-howto.html). See FN 217; some communities use ethics as a criterion of granting recognition to project owners. Communities should adopt by-laws of their umbrella organizations in line with the ethics of community participants.

Towards_an_Improved_def.indd 194 23-02-2010 10:49:35

Apart from taking care of license proliferation and incompatibilities, communities should include in their policies regulations that stimulate licensors not to revoke their licenses. For example, communities could require from some or all contributors to provide additional pledges not to revoke the licenses. Also, copyright assignments to umbrella organizations could be used more often to concentrate copyrights to the project in one entity.[300] It is crucial, however, that such an organization is at the same time legally bound to continue making the project available for the community as Free Software. Otherwise, such a concentration might pose a threat to user freedoms. So, the by-laws of umbrella organizations should explicitly regulate the exercise of copyrights to the project held by the organization and the rules for transferring them from community participants to the organization. For example, the by-laws could bind the organization not to revoke the license or otherwise guarantee that the organization will not misuse the copyrights.

A good licensing policy and institutionalization, such as by incorporating an umbrella organization with sufficiently elaborate by-laws, are necessary conditions of a properly organized software community. But they are not sufficient, and the remaining necessary conditions are non-legal. Namely, appropriate management structure, communication channels, infrastructure for the project, *etc*., should be adopted. Each community should adopt these measures on the basis of their specific needs, while taking into account the teachings of management studies (in particular project management) and the technology currently available.[301] Actually, these non-legal arrangements may prove more efficient in resolving many inefficiencies of the current framework than any legal arrangements. In particular, a community can be properly organized without a formal incorporation of an umbrella organization.

But discussing the non-legal issues further is beyond the scope of this thesis. In this thesis we focus on legal issues, and we analyse non-legal issues only to the extent that it allows us to identify the practical impact that legal rules may have on the freedoms under various circumstances. We should only remind that while designing a community's structure, participants should make it an efficient source of properly working Free Software programs. The organization of the community should stimulate the *community copyleft* mechanism. It should also promote competitiveness in the Free Software market by effectively covering integration costs, by stimulating dis-

300 See: Bruce Byfield, *FSFE's Fiduciary License Agreement is no panacea*, at: http://www.linux.com/feature/60129. See the Fiduciary License Agreement itself at: http://www.fsfeurope.org/projects/ftf/FLA.en.pdf.

301 See *e.g.*, Oliver E. Williamson, *Why Law, Economics, and Organization?*, at: http://papers.ssrn.com/paper.taf?abstract_id=255624; Oliver E. Williamson, *The Economics of Governance*, at: http://www.aeaweb.org/annual_mtg_papers/2005/0107_1645_0101.pdf; Oliver E. Williamson, *The Theory of the Firm as Governance Structure: From Choice to Contract*, 3 Journal of Economic Perspectives 171 (Summer 2002). See also: Giampaolo Garzarelli, *Open Source Software and the Economics of Organization*, at: http://ideas.repec.org/p/wpa/wuwpio/0304003.html.

tributors not to impose restrictive contract on users, by allowing the warranties and services market to operate, as well as by minimizing the likelihood of restricting the freedoms using the architecture.

Here, we may conclude that we end up with the proposal of three improvements that lead to proper organization of software communities.

(1) The communities should continue to elaborate good licensing policies.
(2) The communities should continue to follow the policies, for example by incorporating umbrella organizations that regulate the exercise of project ownership accordingly.
(3) The communities should organize themselves according to the teachings of management studies.

These improvements should be provided for within all communities. We remark that many communities have already introduced these improvements. As far as such communities are concerned we propose that they keep up the good work, while other communities follow their example.

6.2.2 Regulation of eGovernments

For the adequate protection of user freedoms, the introduction of eGovernments should be regulated. Most importantly, governments should be prohibited from introducing Closed eGovernments. To that end, an explicit obligation of governments not to use closed standards should be provided for in the law.[302] If a Closed eGovernment is already operating, governments should be mandated to transform it to an Open eGovernment as soon as possible. It should be made possible to perform this transformation by introducing a Semi-Closed eGovernment. Namely, whenever a closed standard is already used in government communications, or if it is allowed as a narrow exception, governments should be obliged to provide for a translation facility. This boils down to using an open standard substitute with any closed standard used in eGovernment. Only in such a way, the governments could minimize the spill-over effects resulting from Closed eGovernments.

Apart from the above described specific legal obligations, more general, though still legally binding, guidelines for introducing Supra-Open eGovernments should be provided for. Generally, such an extreme Open eGovernment does not have to be mandated,[303] but if the government decides to

302 It could be coupled with narrow exceptions, subject to obligations such as the "comply or explain and commit" obligation. For an explanation of this obligation see: EZ, at: http://www.ez.nl/Onderwerpen/Betrouwbare_telecom/Open_Standaarden_en_Open_Source_Software/Ontwerptekst_instructie_Comply_or_explain_commit (in Dutch).

303 It is advisable to mandate the introduction of Supra-Open eGovernments only if it is proven that the limitations and restrictions of user freedoms cannot be effectively removed using the other improvements that we propose in this chapter.

introduce it, it should be performed diligently. Otherwise, it could affect user freedoms negatively. Most importantly, if governments acquire rights to software, they should be required to make sure the rights allow for its unencumbered development, distribution, and use. Regulation of eGovernments should also cover the performance of the legal audit of licenses by the government. Here, the guidelines can build upon already existing community policies. The government may even collaborate in their drafting with communities active in a given jurisdiction. Government audit of licenses should be transparent, and its results should be made available publicly. Additionally, the guidelines should regulate Free Software licensing performed by the government itself. In particular, the government should be obliged to coordinate any license drafting with already existing community efforts, in order to prevent further proliferation and incompatibilities of licenses. The guidelines should also oblige the government to procure Free Software under a more popular and compatible license if there are two equivalent programs available.

Moreover, guidelines for eGovernments should cover government participation in Free Software development, if the government engages in such a participation. In order to promote proper organization of communities, the government should adhere to such communities that are found to be most properly organized already, or to such communities that promise to organize themselves properly, in particular by showing commitment to follow good licensing policies. So, the government should elaborate criteria for assessment of existing Free Software projects, perform a survey of them, and identify projects maintained by properly organized communities. The criteria should include both legal issues, as well as issues related to project management and the quality of software. Thereafter, the government should examine how it is possible to participate in the identified communities. The governments could also support the *community copyleft* mechanism. For that purpose, the guidelines should lay down procedures for governments to be followed when contributing to community projects.

Furthermore, governments should be obliged to scrutinize contracts with distributors who deliver software to the government. The governments should not be allowed to accept restrictions of user freedoms included in such contracts, if they are not coupled with prevalent benefits. In any case, governments should not accept restrictions which have a spill-over effect on other users. The government should require in particular that the software delivered is free of any architectural restrictions on user freedoms. For an effective removal of the restrictions the government should be allowed to collaborate with the communities in the development and distribution of Free Software. The collaboration may encompass the creation of substitutes to services or devices that restrict the freedoms. Additionally, governments should stimulate the market for warranties and additional services related to Free Software by requiring that Free Software delivered to the government is accompanied with such benefits.

Here, we may conclude that we end up with the proposal of two improvements that lead to the introduction of Open eGovernments.

(1) Governments should be legally prohibited from introducing Closed eGovernments. A swift transformation from existing Closed eGovernments to Open eGovernments should be provided for.
(2) If the government decides to introduce a Supra-Open eGovernment, guidelines should be provided in order to perform this diligently, so that user freedoms are not affected negatively.

The first improvement can be performed by the legislative (both in the terms of providing or clarifying the obligation to use open standards in the applicable law, as well as in the terms of legislating effective measures for the performance and enforcement of such an obligation). The second improvement requires that the government diligently exercises the *dominium* and the *imperium* powers, although it is also possible that such guidelines are provided for in the legislative.

6.2.3 Free Software legislation

Many inefficiencies of the current framework, such as license proliferation and incompatibilities, license revocability, and the *inter partes* nature of licenses can be addressed by the legislative. Precisely speaking, we propose to amend the law if the other improvements that we propose in this chapter fail to remove these limitations and restrictions. Then, the copyright law, contract law, or corporate law could provide for various default terms related to Free Software. For example, if a copyright holder published a program under certain circumstances, users would be allowed to exercise their freedoms as a matter of law. Terms of use provided for in the legislation should be compliant with the FSD and mirror one or some of the most popular model licenses. This could contribute to a decrease of the number of licenses and to an increase of their compatibility. The consequences of the applicability of such a law could be made irrevocable, or the law could allow the copyright holder to claim back the default copyright protection only under specific narrow exceptions. This would affect license revocability.

Also, the freedoms could be turned into *erga omnes* rights as a result of the legislation, which would affect its current *inter partes* nature. How to do it exactly depends on the current status of the applicable law. In any case, adequate protection of user freedoms requires enhancement of the enforceability of *copyleft* clauses. Generally, if the applicable law does not provide adequate standing for licensors and users in *copyleft* enforcement cases, it should be amended to provide for such a standing explicitly. The law should not require that all licensors stand in order to enforce *copyleft*, and it should be allowed for a user to pursue such a case without having to rely on licensors. Users, being the intended beneficiaries of *copyleft*, should be allowed to invoke the same legal claims as licensors (copyright holders). Additionally,

if the applicable law does not contain such a provision, remedies for *copyleft* infringement should explicitly include specific performance, that is the performance of *copyleft* obligations.

Moreover, if in a particular jurisdiction private enforcement of *copyleft* by licensors or users would constitute too much of a burden, other measures should be explored. For example, it should be made clear that *copyleft* infringement can be treated as a breach of copyright, which in many jurisdictions leads to criminal liability.[304] This would allow for a more effective *copyleft* enforcement. However, in order to maintain balance and not impose freedoms on users who do not want them at all, *copyleft* should not be enforced *ex officio*, but only upon a motion of an interested party (a licensor or a user). In other words, either a licensor or a user should be required to initiate any *copyleft* enforcement procedures. The law should also not interfere, if the parties reach a settlement compatible with user freedoms.

Here, we may conclude that we end up with the proposal of two improvements that address inefficiencies such as license proliferation and incompatibilities, license revocability, and the *inter partes* nature of the licenses.

(1) The law should be amended with provisions related specifically to Free Software (such as the ones discussed in this subsection).
(2) Such a law should be enforced by the government using its *imperium*, as well as the judiciary branch.

We propose to implement both these improvements only if the other improvements proposed in this thesis fail to remove the respective inefficiencies of the current framework.

6.2.4 Restriction of software-related patents

Adequate protection of user freedoms requires that software-related patents cannot be used to restrict them. There are patent laws that exclude computer programs from the patentable subject matter in one way or another. The EPC and Polish law are important examples. Also, many stakeholders, not only Free Software advocates, often propose that such provisions are provided for or expanded to make it clear that software-related patents cannot be granted. So, these provisions, where they exist, should be strictly followed. All drafters of amendments to patent law should seriously consider limiting

304 See, *e.g.*, Polish Copyright Act Art. 116, which criminalizes unauthorized distribution of verbatim or modified copies of copyrighted works. A qualified type of this crime is prosecuted *ex officio*, while generally its prosecution requires a motion of the copyright holder (or a collecting society). Distribution of a Free Software program in breach of its license (such as in breach of *copyleft* clause) constitutes unauthorized distribution within the meaning of this provision (especially given the automatic termination in case of a breach provided for in many Free Software licenses).

the patentable subject matter by finding a proper balance between the interests of all stakeholders, including the developers, distributors, and users of Free Software.

Apart from observing or providing for the provisions such as above, the quality of patents should be improved. It should be made more easy to oppose a patent application and to invalidate already granted patents on the grounds that the patentability criteria are not met.[305] Also, patent offices should be equipped with sufficient resources to be able to scrutinize incoming applications diligently. Patent laws that allow to grant patents without performing any scrutiny of claimed inventions, or which allow that only a minimal scrutiny is performed, should not stand. Obviously, many patent offices will not be able to perform a thorough scrutiny of all applications, which are being filed in an increasing number. So, the offices should be encouraged to collaborate with each other to exchange patent information and expertise. The collaboration should be expanded to allow involvement of experts from other patent offices (or maybe even from the general public) in the review of applications and in submitting prior art. In particular, patent offices should closely observe community efforts related to patents and support them where possible.[306] As a result of gathering and combining patent-related information from various sources, examiners would be able to scrutinize applications more diligently.

We indicate that patent holders themselves are most fit for identifying that their patents are infringed. So, it is advisable to introduce a legal obligation for patent holders to notify their patents to alleged infringers, if they want to be allowed to enforce these patents. In other words, enforceability of a patent should be conditioned upon such a notification. For this purpose, a special repository of software should be established.[307] Projects submitted to such a repository would benefit from the following safe harbour regulation. Within a certain period after a project (or its improvement) is submitted, every patent holder would be required to notify any infringement alleged in relation to the project.[308] After notification, the project would be given an additional grace period for developing a workaround or to remove the patented technology completely. If the infringement is not notified, or if the workaround is not objected by the patent holder, there should be a statute of limitations on the enforceability of the patents in question against every

305 WIPO (Standing Committee on the Law of Patents), *Report on the International Patent System*, (3 February 2009, SCP/12/3 Rev. 2), p. 56-57.

306 WIPO (Standing Committee on the Law of Patents), *Report on the International Patent System*, (3 February 2009, SCP/12/3 Rev. 2), p. 54.

307 This repository does not have to be restricted to Free Software only.

308 Holders of newly granted patents could be given longer time in order to scrutinize all projects present in the repository to date. At the same time applicants for new patents should be obliged to perform an analysis of the repository before submitting an application and indicating how their claimed invention differs from the programs found in the repository in order to allow for more effective review of the applications. Such a repository would constitute an important source of prior art.

Towards_an_Improved_def.indd 200 23-02-2010 10:49:36

developer, distributor, or user of the project.[309] Namely, the patent holder (1) would not be allowed to obtain an injunction against them, or (2) would not be allowed to claim any royalties or damages, or (3) both.[310]

Within such a safe harbour, conciliation or arbitration procedures should be performed in case the proposed workaround is not found satisfactory for patent holders, or if someone questions the applicability of the notified patent towards a particular project. In case the parties find it helpful, negotiations should be supported by a mediation in order to allow them to agree on a patent license which would be satisfactory from the point of view of user freedoms. Thereafter, the licensed patent should be included in a patent pool available for any interested party who would like to use it in relation to Free Software. For that purpose, model patent licenses should be elaborated.[311] It should be discussed whether such licenses could provide for patent royalties. Indeed, it may be the case that not all royalties are incompatible with user freedoms. For example, a capped sum depending on revenues might allow to exercise user freedoms without major problems.[312]

Here, the question arises, how to provide for such a safe harbour in practice? In order to be effective, the safe harbour should be based on a firm legal basis. In order to affect patents worldwide, the legislatives of all jurisdictions should act together. Definitely, such a worldwide patent law reform will not be an easy task.[313] But it may suffice for some major jurisdictions to perform

309 This proposal is similar to the already existing "notice and take-down" regulations, existing under the DMCA Sec. 512, the E-Commerce Directive, and in the Directive's national implementations. Instead of a statute of limitations, a license of right could be provided for.

310 Similar limitation of rights of patent holders has been endorsed by the IBM in their contribution towards the discussion about the European Community Patent. It was developed after one of the proposals made by EPO and called "Soft IP" (See: IBM, *The European Community Patent revisited Discussion paper from IBM*, at: http://www.epip.eu/conferences/epip02/lectures/European%20Interoperabily%20Patent%201.1.pdf).

311 See the GPLv3 and its patent clause.

312 See: David Worthington, *TomTom can license FAT without violating GPL*, at: http://www.sdtimes.com/blog/1364; see also: David Worthington, *Experts: Microsoft's FAT licensing terms might violate GPL*, at: http://www.sdtimes.com/link/33327.

313 The discretion of national legislatives to limit patents are subject to such provisions as the Paris Convention Art. 5A or TRIPS Arts. 30 and 31. The Paris Convention Art. 5A would apply if the safe harbour is found to be an abuse-prevention measure. Then, in order to comply with that provision, the safe harbour should result in compulsory licenses rather than other measures (Art. 5A.3). A compulsory licensing system effective towards all users of Free Software might prove problematic to implement. However, the safe harbour may be provided for as an exception subject to TRIPS Art. 30 or 31 (*i.e.*, not as an abuse-prevention measure). A statute of limitations on the enforcement of patents towards a specific repository of software does not seem to fall outside Art. 30. Also, in order to escape from the strict requirements of Art. 31, the safe harbour should focus on measures stimulating unencumbered negotiation of patent licenses, as well as facilitating workarounds of patents. Only if the above fails, the use of patents should be subject to authorizations granted in conformity with Art. 31, possibly with the use of competition law institutes.

it, in order for the rest to follow. For example, the current (2009) discussion about the European Community Patent can provide a good forum for presenting the concept of the safe harbour. Namely, the safe harbour can be implemented at the pan-European level without affecting national patent systems. The Community Patent system could give patent holders a choice whether to pursue the Community-wide patent subject to the safe harbour, or to apply for national patents that would not be subject to such a regulation for the time being.[314]

In the meantime, before the law is amended, the necessary prerequisites for the safe harbour can be created by the communities using private ordering. For that purpose, the communities should cooperate in the setting of the repository described above (or in the transformation of the already existing Free Software repositories accordingly). They should also draw the attention of patent offices to it, in particular to make the repository an official source of prior art. Additionally, the communities should attempt to gather a critical mass of patent holders that contribute their patents to users of software.[315]

Here, we may conclude that we end up with the proposal of three improvements of software-related patents.

(1) Patent laws should be reformed to limit the scope of patentability of software or to provide that already existing exclusions of software from the patentable subject matter are strictly followed.
(2) Regardless of the reform, patent quality should be improved.
(3) A safe harbour should be established by appropriate legislation in order not to allow enforcement of patents against user freedoms.

These improvements can be implemented by the legislative, but involvement of the government using its *imperium* and *dominium* powers would most probably be necessary as well. The communities should become involved too.

6.2.5 Promotion of open standards

Adequate protection of user freedoms requires that open standards are promoted. This implies improvements directed at SSOs. Most importantly, SSOs

314 It has been proposed to implement the IBM's proposal in much the same way. (See: IBM, *The European Community Patent revisited Discussion paper from IBM*, at: http://www.epip.eu/conferences/epip02/lectures/European%20Interoperabily%20Patent%201.1.pdf)

315 Patent holders who gather so-called "defensive patent portfolios" should be interested in creating defensive patent pools together with other such patent holders. At the same time, patent holders who use their patents offensively might be interested in the existence of a repository where a big number of potential targets of their claims could be easily located. But even if it would make it easier for them to pursue alleged infringements, the information about prior art and the support of all participants in the repository might also make it easier for alleged infringers to defend themselves effectively.

Towards_an_Improved_def.indd 202 23-02-2010 10:49:37

should make sure that the standard setting procedure is open to all interested parties. They should also amend their internal regulations to provide for an explicit obligation of participants not to restrict standards. This prohibition should refer to trade secrets, scrambling of interoperability information, and lock-ins, but most importantly it should cover patents. For that purpose, SSOs should diligently define what patents they consider to be material to standards. Participants should be obliged to notify any such patents (and pending applications). SSOs should also require that the participants offer patent licenses compatible with the freedoms. If an SSO fails to develop an open standard despite the above improvements, this fact should be communicated to the public in order to make everyone aware that the standard is closed.

SSOs that strictly adhere to the above recommendations might not be attractive for all market players. Also, SSOs cannot easily bind third parties, who do not agree to follow the internal rules of SSOs. For these reasons, a safe harbour for standards should be provided for by the legislative, in a similar manner as discussed above in relation to software-related patents. Namely, patent holders should be obliged to review standards of selected SSOs (or a repository of standards could be organized). They should be required to notify any patents they find material to standards being designed within certain deadline. Without such a notification or after the deadline passes, enforcement of these patents should not be allowed. The safe harbour should also stimulate licensing of patents under terms compatible with the freedoms, and creation of patent pools available for any party interested in the exercise of the freedoms. It is additionally advisable to propose certification of the licenses, in a similar manner to certification (scrutiny) of software licenses performed by the OSI or the FSF. At the same time, exceptions from the rights of patent holders should be provided for if a patent is infringed in order to provide for interoperability.[316]

The above improvements should be carefully implemented in order not to remove incentives that stimulate many stakeholders to participate in a standard setting. As we already observed, future patent royalties are an incentive for some stakeholders. Although royalties might not be in conflict with freedoms, the conditions imposed by patent holders are. So, while providing for these improvements a substitute monetary incentive should be provided for and the patent royalties incentive should be removed. This can be performed by governments when introducing eGovernments. Most importantly, governments should be obliged to use multi-purpose open standards whenever possible. This would stimulate a wider use of open-standards-based software without causing a risk of a lock-in to any particular vendor. It would contribute to a creation of a market for products of many vendors. Government demand, and the spill-over effect it has on the overall

316 Such exceptions can be similar to the decompilation right already present in the Software Directive or other software copyright regulations.

market demand, would promise vendors a return on investment, a monetary incentive. But it is crucial that after a given standard is chosen for eGovernment, the law should limit the enforceability of patents towards anyone that uses the standard. Otherwise, patent holders would benefit both from government demand and from patents, which would have a destructive effect on user freedoms.

But the government can promote open standards in many other ways too. Governments can increase their awareness to standard setting, as well as their participation in it. They may require SSOs to follow procedures that lead to open standards. Government involvement does not have to be limited to standards used in government communications. Also, the legislative can extend the applicability of internal rules of SSOs to non-participating parties. Additionally, the government should stimulate more cooperation between patent offices and SSOs. They should be required to exchange information, which could help patent offices to identify prior art, while at the same time it could help the SSOs to avoid designing standards encumbered with patents.

Here, we may conclude that we end up with the proposal of three improvements of the standard setting.

(1) Subject to the above reservations, SSOs should be reformed to guarantee that they design open standards only.
(2) A safe harbour for standards should be established, in a manner similar to the safe harbour for software discussed in the previous subsection.
(3) The government should stimulate the design and popularity of open standards in particular by selecting multi-purpose open standards for eGovernments and by limiting enforceability of patents with regard to standards used in eGovernment.

The first improvement can be provided for by the communities. The communities should increase their participation in SSOs and initiate the necessary reform. The first improvement can be introduced by governments simultaneously, since governments often participate in SSOs. Additionally, the governments could stimulate SSOs by using their *imperium*, as well as the legislative. The other two improvements can be performed by the legislative, as well as by a diligent exercise of the *imperium* and *dominium* powers of the government.

6.2.6 Internationalization of the framework

For an adequate protection of user freedoms, any differences between national laws should not be allowed to restrict them. So, the legislatives of each jurisdiction should scrutinize their laws and undertake harmonization efforts. This can encompass including Free Software issues in the agendas of such supra-national organizations as the EU, WIPO, and WTO. But legisla-

tive procedures at the national and international level are time consuming. In the meantime, the communities could provide for the necessary temporary improvements. Amending licenses to make them more independent of particular jurisdictions (such as the GPLv3 as compared to GPLv2) should be promoted. Also, if a particular jurisdiction is found to protect the freedoms better than others, the licensors should be stimulated to use choice-of-law and choice-of-forum clauses which point to such a jurisdiction. Nevertheless, license drafting and forum shopping cannot guarantee that the applicability of a particular national law is avoided. Many laws contain mandatory rules and many courts can find their jurisdiction despite a choice-of-forum clause (or if such a clause is invalidated).

So, it is advisable to explore additional measures that could deal with applicable laws that still contain restrictions of user freedoms. This can be made by treating the regulatory framework of Free Software as *lex mercatoria*.[317] *Lex mercatoria* is usually understood as a system of rules and relations between the rules that is independent of any specific national law. It is believed to be applicable in a uniform way across jurisdictions. Such a system is in particular capable of circumventing restrictions that follow from an applicable law. If a court is allowed to apply *lex mercatoria*, it either does not have to follow its national law strictly, or it may interpret the national law by accounting for a specific rule from the *lex mercatoria*.[318] Indeed, nowadays the concept of *lex mercatoria* has been often used in the international trade, especially in disputes where none of the parties could accept the applicability of any particular national legal system.[319]

Free Software licensing and project ownership as exercised within the communities have created a set of customs and trade practices that can be treated as *lex mercatoria*. But mere existence of such a *lex mercatoria* does not make national laws inapplicable. The courts would not apply any such rules instead of national legal rules, unless there is an explicit regulation that allows or requires them to do so. We are aware of such laws that allow courts to refer to equity, or allow them to account for custom, trade practices, *etc*. If a particular applicable law does not contain such provisions, it should be amended accordingly. In any case, courts should be given clear guidelines

317 Wikipedia, *Law Merchant*, at: http://en.wikipedia.org/wiki/Lex_mercatoria#The_medieval_Law_Merchant. For an interesting critique of the existence of *lex mercatoria* in the medieval times see: Stephen Edward Sachs, *From St. Ives to Cyberspace: The Modern Distortion of the Medieval 'Law Merchant'* 21 American University International Law Review 685 (2006), also available at: http://ssrn.com/id=830265. For various other definitions of *lex mercatoria* see, *e.g.*, Bernadetta Fuchs, Lex mercatoria w międzynarodowym obrocie handlowym [Lex mercatoria in international trade], 17 *et.seq.* (Zakamycze, 2000) (in Polish).

318 See: Przemysław P. Polański, Customary Law of the Internet (TMC Asser Press 2007) 111 *et seq.*, 347 *et. seq.*

319 For a far analogy in criminal court decisions see, *e.g.*, F. O. Raimondo, General Principles of Law in the Decisions of International Criminal Court and Tribunals (2007, Ph.d. thesis at University of Amsterdam).

that when deciding cases related to Free Software, due account of the *lex mercatoria* should be performed. *Lex mercatoria* can be more easily applied before arbitral tribunals. So, the communities should consider exploring arbitration of Free-Software-related disputes. However, there are limitations of this direction, since arbitration is possible if only parties agree, and usually it is required that arbitration clauses and agreements are executed in writing.[320]

Here, we may conclude that we end up with the proposal of three improvements of the regulatory environment.

(1) Free Software issues should be included into agendas of international organizations that are able to stimulate harmonization of national laws.
(2) Before the international harmonization is performed, Free Software licenses should be amended to become more independent of particular jurisdictions in order to stimulate their uniform application.
(3) Courts should be allowed and required to look into customs and trade practices developed in the Free Software scene when deciding on disputes related to user freedoms. Treating the regulatory framework as lex *mercatoria* could be one way of reaching this goal.

All these improvements require a cooperation between the government (acting through all its branches) and the communities.

6.3 *Construction of a proposal of an improved framework*

In this section we use the sixteen improvements proposed in the previous section for the construction of an improved regulatory framework of Free Software. Above, we grouped the improvements in the following six classes: (1) proper organization of the communities (three proposed improvements), (2) regulation of eGovernments (two proposed improvements), (3) Free Software legislation (two proposed improvements), (4) restriction of software-related patents (three proposed improvements), (5) promotion of open standards (three proposed improvements), and (6) internationalization of the framework (three proposed improvements).

Each of the classes includes two or three specific improvements, all of which have been explained in the previous section. Together, the proposed improvements address all limitations and restrictions of the freedoms (the inefficiencies of the current framework) discussed throughout the thesis. Strictly speaking, the implementation of the proposed improvements can affect the inefficiencies so that the impact of the inefficiencies on the rules that protect the freedoms is minimized, or the inefficiencies are removed

320 New York Convention Art. II.

completely. We speculate on the provisional impact of our proposed framework in Section 7.3.

In Figure 6.1 we present the complete proposal of our improved regulatory framework. This framework includes all rules included in the model, and all relations between the rules (see Figure 3.9). It also includes all improvements that we proposed in this chapter.

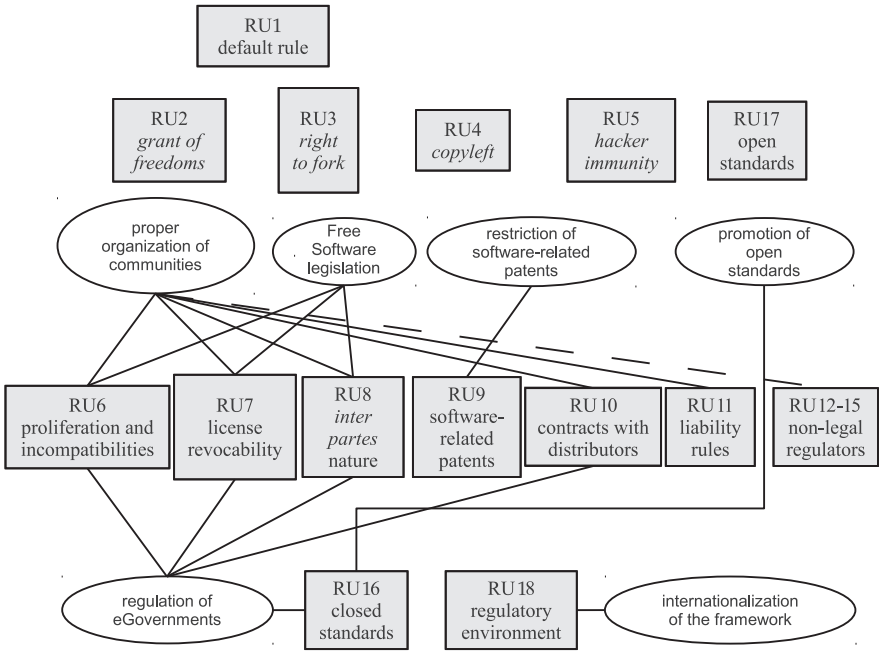

Figure 6.1: Our improved framework as proposed in Chapter 6

For the sake of clarity, in Figure 6.1 we present only the relations between the improvements and the inefficiencies. We do not present all relations between the rules existing in the current framework that we already identified in Chapter 3.

6.4 Chapter summary

The model of the current framework that we reconstructed in this thesis (see Chapter 3) includes *the default rule* and many other rules. The other rules can be divided into two kinds. Rules of the first kind grant users their freedoms and attempt to protect the freedoms. They are based on *the default rule*, which gives exclusive control over programs to their copyright holders. These are: (1) *the grant of freedoms*, (2) *the right to fork*, (3) *copyleft*, and (4) *the hacker immunity*. Rules of the second kind limit or restrict the freedoms. They cause the inefficiencies of the current framework. These are: (1) license proliferation

and incompatibilities, (2) license revocability, (3) *inter partes* nature of the licenses, (4) software-related patents, (5) contracts with distributors, (6) liability rules, (7) non-legal regulators, (8) closed standards, and (9) the regulatory environment.

The current framework operates in the world of software communities and eGovernments. Both the communities and eGovernments affect the framework. The communities are able to overcome or minimize many of the inefficiencies, but only if they are properly organized. However, even properly organized communities are unable to overcome all inefficiencies, while many of them are only avoided, not eliminated by the communities. Then, eGovernments can affect the framework in both directions and sometimes they are neutral towards the framework. This depends on how eGovernments are introduced precisely. Our analysis of the operation of the framework in the world of software communities and eGovernments has (re)established the conclusion that under the current framework user freedoms are not sufficiently protected.

In this chapter, we discussed various improvements of the framework and we proposed an improved framework. The improved framework consists of improvements that we selected as most appropriate. We grouped the improvements under the following classes: (1) proper organization of the communities, (2) regulation of eGovernments, (3) Free Software legislation, (4) restriction of software-related patents, (5) promotion of open standards, and (6) internationalization of the framework. The model of the current framework, amended with all the improvements constitutes our proposal of an improved framework.

So, we provide the following answer the RQ3, which was "*How to improve the regulatory framework so that it adequately protects user freedoms, as articulated by Stallman, in the world of software communities and eGovernments?*"

In order to improve the regulatory framework so that it adequately protects user freedoms in the world of software communities and eGovernments, the framework should be amended with all improvements proposed in this chapter in the manner discussed in detail hereto. We briefly reference all improvements below.

(1) **Proper organization of communities.** The communities should continue to elaborate good licensing policies. The communities should also continue to follow the policies, for example by incorporating umbrella organizations that regulate the exercise of project ownership accordingly. Additionally, the communities should organize themselves according to the teachings of management studies.

(2) **Regulation of eGovernments.** The law should prohibit governments to introduce Closed eGovernments. A swift transformation from existing Closed eGovernments to Open eGovernments should be provided for. If the government decides to introduce a Supra-

Open eGovernment, guidelines should be provided in order to perform this diligently, so that user freedoms are not affected negatively.

(3) **Free Software legislation.** If the other improvements proposed in this thesis should fail to protect the freedoms adequately, the law should be amended with provisions related particularly to Free Software (*i.e.*, default terms of use that mirror popular Free Software licenses, possibly irrevocable, and *copyleft* obligations effective *erga omnes*).

(4) **Restriction of software-related patents.** Patent laws should be reformed to limit the scope of patentability of software or to provide that already existing exclusions from the patentable subject matter are strictly followed. Also, patent quality should be improved. Additionally, a safe harbour should be established by appropriate legislation and community efforts in order not to allow enforcement of patents against user freedoms.

(5) **Promotion of open standards.** SSOs should be reformed to guarantee that they design open standards only. Also, a safe harbour for standards should be established, in a manner similar to the safe harbour that would restrict software-related patents. Additionally, the government should stimulate the design and popularity of open standards in particular by selecting multi-purpose open standards for eGovernments and by limiting enforceability of patents with regard to standards used in eGovernment.

(6) **Internationalization of the framework.** Free Software issues should be included into agendas of international organizations that are able to stimulate harmonization of national laws. Before the harmonization is performed, licenses should continue to be amended to become more independent of particular jurisdictions, in order to stimulate their uniform application. Also, courts should be allowed and required to look into customs and trade practices developed in the Free Software scene when deciding disputes related to user freedoms. Treating the regulatory framework as lex mercatoria could be one way of reaching this goal.

Towards_an_Improved_def.indd 210 23-02-2010 10:49:37

7 Conclusions and further research

In this chapter we complete our research by answering the Research Questions (in Section 7.1) and the Problem Statements (in Section 7.2). In Section 7.3 we speculate on the provisional impact of our proposed framework. We also identify issues for further research (in Section 7.4).

7.1 Answers to research questions

In this thesis we formulated three Research Questions. In the course of our analysis, we answered all RQs. The answers were presented in Chapters 4, 5, and 6, respectively. The answers were formulated at the end of each chapter as a summary of more detailed findings presented throughout the analysis. In the thesis we also presented many intermediate conclusions that should be read together with the answers and findings. Below, we highlight the essence of the answers in the summary form. For details, we refer the reader to the extended answers given at the end of corresponding chapters.

RQ 1: *In what way do software communities affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?*

Software communities affect the current regulatory framework concerning the protection of user freedoms in the following nine ways.

(1) **License proliferation and incompatibilities.** Properly organized communities are able to perform the legal audit necessary to overcome license proliferation and incompatibilities between Free Software licenses.
(2) **License revocability.** The fact that a program is properly maintained by a community may simply make licensors unwilling to revoke licenses because of the benefits of community maintenance.
(3) **Inter partes nature of licenses.** Properly organized communities are able to provide for the *community copyleft* stimulation. This minimizes the limitation that follows from the inter partes nature of the licenses.
(4) **Software-related patents.** Some properly organized communities stimulate some patent holders not to enforce their patents. However, software-related patents remain an important threat in the world of software communities.

(5) **Contracts with distributors.** Properly organized communities prevent distributors to impose too restrictive contracts on users. Alternatively, such communities constitute a substitute source of software available without contractual restrictions.
(6) **Liability rules.** Properly organized communities provide for an effective maintenance which leads in particular to a reduction of the number of defects in programs. This minimizes the negative effect that liability rules may have on user freedoms.
(7) **Non-legal regulators.** Properly organized communities perform quality audit or otherwise control the development of Free Software so that the software is maintained on an ongoing basis.
(8) **Closed standards.** Properly organized communities may contribute to a slowdown in adoption of closed standards. Some communities may even decrease incentives for the designers to make closed standards.
(9) **Regulatory environment.** As any other entities, communities operate in a regulatory environment where many national laws have to be taken into consideration. But the exercise of the freedoms in the communities is regulated by customs and trade practices to a material extent.

RQ 2: *In what way do eGovernments affect the current regulatory framework concerning the protection of user freedoms, as articulated by Stallman?*

eGovernments affect the current regulatory framework concerning the protection of user freedoms in the following four ways.

(1) **Closed eGovernments.** Closed eGovernments affect the framework negatively from the point of view of user freedoms. Precisely speaking, they affect *the grant of freedoms*, liability rules, non-legal regulators, and closed standards. Given the likelihood of the spillover effect, these rules are affected with regard to software used in government communications, as well as to other software.
(2) **Semi-Closed eGovernments.** If the government proceeds diligently, it may succeed in the introduction of a Semi-Closed eGovernment. Such an eGovernment enables users to exercise user freedoms despite the fact that closed standards are used in government technologies. But given its limitations it should be taken under consideration only as a transitional eGovernment from a Closed eGovernment to an Open eGovernment.
(3) **Open eGovernments.** Open eGovernments are capable of avoiding the negative effects on the framework that are a result of Closed eGovernment. Still, they do not affect most of the rules, they are merely neutral towards them. Open eGovernments only promote open standards as compared to closed standards, and they do not remove closed standards from the framework.

(4) **Supra-Open eGovernments.** The government is capable of affecting the framework to a greater extent by introducing a Supra-Open eGovernment. However, the exact outcome of such an eGovernment depends much on how it is introduced precisely. Also, even Supra-Open eGovernments are not able to overcome such limitations and restrictions as license revocability, *inter partes* nature of the licenses, software-related patents, liability rules, closed standards, and the regulatory environment. Affecting these limitations and restrictions is possible if only apart from the *dominium*, the government additionally employs the *imperium* or other branches (the legislative and the judiciary).

RQ 3: *How to improve the regulatory framework so that it adequately protects user freedoms, as articulated by Stallman, in the world of software communities and eGovernments?*

The current regulatory framework should be improved by implementing sixteen improvements belonging to the following six classes of improvements, so that the resulting new framework adequately protects user freedoms in the world of software communities and eGovernments.

(1) **Proper organization of communities.** The communities should continue to elaborate good licensing policies. The communities should also continue to follow the policies, for example, by incorporating umbrella organizations that regulate the exercise of project ownership accordingly.
(2) **Regulation of eGovernments.** The law should prohibit governments to introduce Closed eGovernments. A swift transformation from existing Closed eGovernments to Open eGovernments should be provided for.
(3) **Free Software legislation.** If the other improvements proposed in this thesis should fail to protect the freedoms adequately, the law should be amended with provisions related particularly to Free Software (*i.e.*, default terms of use that mirror popular Free Software licenses, possibly irrevocable, and *copyleft* obligations effective *erga omnes*).
(4) **Restriction of software-related patents.** Patent laws should be reformed to limit the scope of patentability of software. A safe harbour should be established in order not to allow enforcement of patents against user freedoms.
(5) **Promotion of open standards.** SSOs should be reformed to guarantee that they design open standards only. Also, a safe harbour for standards should be established, in a manner similar to the safe harbour that would restrict software-related patents.
(6) **Internationalization of the framework.** Free Software issues should be included into agendas of international organizations that

are able to stimulate harmonization of national laws. Before the harmonization is performed, licenses should continue to be amended to become more independent of particular jurisdictions, in order to stimulate their uniform application. Also, courts should be allowed and required to look into customs and trade practices developed in the Free Software scene when deciding disputes related to user freedoms. Treating the regulatory framework as *lex mercatoria* could be one way of reaching this goal.

7.2 Answers to the problem statement

In this thesis we formulated a twofold Problem Statement. In order to answer the Problem Statement we formulated a Research Goal and three Research Questions. The Research Goal is to develop a new, improved regulatory framework that would be capable of resolving the inefficiencies in the protection of user freedoms identified by analysing the Problem Statement. In the previous section we presented answers to the three Research Questions that led us to the proposal of an improved regulatory framework. By formulating the proposal of an improved framework presented in Chapter 6, we reached the Research Goal of the thesis. We also answered the Problem Statement, and we presented each part of the twofold answer in Chapters 4 and 5 respectively. We reproduce these answers below in brief.

PS 1: *What are the relations between user freedoms and software communities?*

Software communities enter into relations with all nine inefficiencies (*i.e.*, limitations and restrictions) that we included in the model of the current framework. Properly organized software communities materially minimize the inefficiencies, in a manner described in more detail in Chapter 4. As a result, such communities help users to exercise the freedoms in practice. However, there are three inefficiencies that remain an important threat of user freedoms, despite the communities. These are: (1) software-related patents, (2) closed standards, and (3) the regulatory environment.

PS 2: *What are the relations between user freedoms and eGovernments?*

There are three main relations between user freedoms and eGovernments.

(1) Closed eGovernments are capable of restricting user freedoms effectively, despite the protection provided for in the framework, and despite the positive effect on the freedoms that properly organized communities may have.
(2) Semi-Closed eGovernments enable users to exercise their freedoms despite the fact that closed standards are used by the government.

But their introduction requires much diligence. Still, such eGovernments do not remove the limitations and restrictions to which users are exposed in a world without eGovernments.

(3) Open eGovernments are capable of avoiding the additional restriction of user freedoms that is a result of Closed eGovernments. But they are mostly neutral towards the framework, although they materially affect closed standards. Only Supra-Open eGovernments are able to minimize all existing limitations and restrictions materially. Similarly to Semi-Closed eGovernments, the introduction of Supra-Open eGovernments also requires much diligence. However, any Open eGovernment is still unable to remove all existing limitations and restrictions of user freedoms, unless *the imperium* or the legislative and the judiciary becomes involved.

7.3 Provisional impact of our proposed framework

Our proposed framework consists of all rules and relations existing in the current framework, and of the sixteen improvements that we proposed to implement. The implementation of the improvements can lead to two consequences. First, as a result of the implementation new rules are created that substitute the rules that caused the inefficiencies. This means that the latter rules are completely removed. Second, the implementation leads to an introduction of new rules that will exist in a framework together with the rules that lead to the inefficiencies. This means that the rules that lead to the inefficiencies will not be removed completely, but they will be in a relation with the rules introduced as a result of the improvements. Naturally, for an adequate protection of user freedoms such a relation should lead to a minimization of the inefficiencies to the largest possible extent.

Whether the implementation of our proposed improvements leads to the consequences of the first or of the second kind depends on how the implementation is performed. The final effect of the improvements requires further research, most probably empirical research. Here, we are only able to speculate about the provisional impact of our proposed framework. Below, we analyse (1) what rules could be created if the improvements are to be implemented, and (2) how these new rules would impact the rules existing in the current framework. Then, we present (3) our speculative conclusions on the provisional impact of our proposed framework.

7.3.1 Rules created if the improvements are implemented

In Chapter 6 we proposed sixteen improvements in the current framework and we grouped them in the following six classes: (1) proper organization of communities, (2) regulation of eGovernments, (3) Free Software legislation, (4) restriction of software-related patents, (5) promotion of open standards, and (6) internationalization of the framework. Below, we attempt to identify

the rules that could be created if each of the improvements is to be implemented.

The proper organization of communities class consists of the following three improvements: (1) elaboration of licensing policies, (2) following the policies, and (3) organization according to the teachings of management studies. All these improvements can be undertaken by the communities, without engaging the government. Communities are able to regulate their participants, not third parties. They may do so using the law, but they may also regulate using such non-legal regulators as the norms. If the above improvements are introduced using the law, the communities will include licensing policies and organization in their by-laws, such as the by-laws of umbrella organizations. So, we may expect that the following legal rule could be created as a result of the implementation of all three improvements included in the proper organization of communities.

> **Licensing policies.** Project owners are *not allowed* to accept (to the official version of the project) contributions that do not comply with the project's licensing policy.

We remark that the above rule has already been introduced in communities that have elaborated licensing policies and organized themselves accordingly. We also remark that there may be other rules (legal rules or norms) that could result out of proper organization of communities. Given the fact that these rules depend on particular communities, it is not possible to identify them here. Determination of these rules calls for further research on communities and their organization.

The regulation of eGovernments class consists of the following two improvements: (1) a legal prohibition to introduce Closed eGovernments, and (2) an obligation to diligently introduce a Supra-Open eGovernment. The first improvement can be performed by the legislative. The second improvement requires that the government diligently exercises the *dominium* and the *imperium* powers, although it is also possible that such guidelines are provided for in the legislative. Both these improvements boil down to imposing certain legal obligations on governments (public administrations). So, we may expect that the following two legal rules could be created as a result of the implementation of the improvements included in the regulation of eGovernments.

> **Prohibition of Closed eGovernments.** The government is *not allowed* to use closed standards and is obliged to use open standards.

> **Guidelines for Supra-Open eGovernments.** If a decision to introduce a Supra-Open eGovernment is made, it *should not* cause or support inefficiencies in the protection of user freedoms.

The Free Software legislation class consists of the following two improvements: (1) amending the law with provisions related specifically to Free Software,

and (2) enforcing such a law. We remark that we proposed to implement these improvements if other improvements proposed in this thesis should fail to provide for an adequate protection of user freedoms. In such a case, Free Software legislation is able to address many inefficiencies of the current framework in many different ways. So, it may result in many different new rules in the framework. Let us provide an example. Namely, if it is found that the *inter partes* nature of licenses has to be regulated in the law in order to allow for better enforceability of *copyleft*, a following legal rule could be created.

> **Extended *copyleft* enforceability.** Every user is *allowed* to enforce obligations of the licensees under *copyleft*.

The restriction of software-related patents class consists of the following three improvements: (1) a reform of patent laws, (2) an improvement of patent quality, and (3) a safe harbour for user freedoms. The rules that will follow from the first two improvements will differ depending on the actual patent law that is improved as well as on the balance reached between the interests of all stakeholders. As a result, many different new rules may appear in the framework, each limited to affect a given jurisdiction, so that all of them together provide for an adequate protection against software-related patents worldwide. Drafting of all such rules requires further research. Here, we can perform a provisional identification of a legal rule that may result out of the third improvement, that is of a safe harbour against software-related patents. It could be the following legal rule.

> **Safe harbour against software-related patents.** Users are *allowed* to exercise all six activities covered in the FSD despite such activities constitute an infringement of a patent, under condition that the patent holder failed to meet patent enforceability criteria specified by the safe harbour provisions applicable to the program in question.

The promotion of open standards class consists of the following three improvements: (1) reform of the SSOs, (2) a safe harbour for standards, and (3) government stimulation of open standards. The rules that will follow from the first improvement are highly dependent on the particular SSO that is going to be reformed. So, many different new rules may appear in the framework as a result of reforming many different SSOs. The second improvement can be expected to result in a legal rule similar to the above described rule that will result from the safe harbour against software-related patents. Namely, we may expect that the following rule could be created.

> **Safe harbour against patents material to standards.** If compliance with a standard requires the use of a subject-matter covered by a patent, users of programs that use the standard are *allowed* to exercise activities covered by the FSD, under condition that the patent holder failed to meet patent enforceability criteria specified by the safe harbour provisions applicable to the standard in question.
> In particular, developers are *allowed* to develop Free Software that uses such a standard, and distributors are *allowed* to distribute this software.

The third improvement included in the *promotion of open standards* class can be expected to result in a legal rule that would be a refinement of the above rule named "prohibition of Closed eGovernments". Namely, the following rule could be created.

> **Prohibition of Closed eGovernments (refinement).** The government is *not allowed* to use closed standards and is obliged to use multi-purpose open standards.

Alternatively, the following rule could be created as a result of the third improvement included in the *promotion of open standards* class.

> **Limited enforceability of patents material to standards.** If compliance with a standard requires the use of a subject-matter covered by a patent, users of programs that use the standard are *allowed* to exercise activities covered by the FSD with regard to these programs if the standard has been offered and accepted for the use in an eGovernment.
> In particular, developers are *allowed* to develop Free Software that uses such a standard, and distributors are *allowed* to distribute this software.

The internationalization of the framework class consists of the following three improvements: (1) inclusion of Free Software issues into agendas of international organizations, (2) amendment of Free Software licenses, and (3) requiring courts to take customs and trade practices of the communities into consideration when deciding on disputes related to user freedoms. There is no specific rule that can be construed as a result of the first two improvements. Rather, these improvements are a means for implementing other improvements proposed in this thesis. For example, an international organization can prepare a model Free Software legislation or it may serve as a forum for discussing the exact rules for a safe harbour against software-related patents. As far as the third improvement is concerned, courts serve as means for implementing (applying) existing legal rules. Also, a legal rule requiring courts to look into customs and trade practices of software communities will also be highly dependant on the court procedure of the particular jurisdiction that will implement the improvement. It requires more research to identify how such rules will be drafted in particular jurisdictions. Here, we cannot provide any description of the rules that could follow from the improvements included in the internationalization of the framework class.

Here, we may conclude that we expect at least eight rules to be created as a result of our sixteen proposed improvements. One of these eight rules (prohibition of Closed eGovernments (refinement)) is a refinement of another of them (prohibition of Closed eGovernments). The former includes the latter, so we will further discuss only the former, which gives seven new rules altogether. It should be expected that the final number of rules will be higher than seven, since here we are only analysing the provisional impact of our improved framework. The exact number and the formulation of all these additional new rules is subject to many circumstances that are beyond the scope of this thesis. Further research, in particular empirical research, is necessary in order to do so.

7.3.2 Impact of the new rules on the existing rules

Below, we analyse how the new rules that we identified in the previous subsection may impact the rules existing in the current framework. We will analyse the impact of the following seven new rules: (1) licensing policies, (2) prohibition of Closed eGovernments (refinement), (3) guidelines for Supra-Open eGovernments, (4) extended copyleft enforceability, (5) safe harbour against software-related patents, (6) safe harbour against patents material to standards, and (7) limited enforceability of patents material to standards.

The *licensing policies* rule is a result of the improvements belonging to the *proper organization of communities* class. So, it will have an impact on the following six inefficiencies of the current framework: (1) license proliferation and incompatibilities, (2) license revocability, (3) *inter partes* nature of the licenses, (4) contracts with distributors, (5) liability rules, and (6) non-legal regulators. The licensing policies cannot regulate actors outside of the communities and they should not be expected to have too strong an impact on community members either. So, it is unlikely that the licensing policies will completely remove any of the above inefficiencies from the framework. But although these inefficiencies will remain in the framework, we can expect that their negative impact on the freedoms would be minimized due to properly organized communities, as this is the case in such communities that already operate. Members of properly organized communities will be the direct beneficiaries. It should be also expected that as projects that are developed and distributed by properly organized communities become popular, the above mentioned inefficiencies will stop to affect many other users materially as well.

The *prohibition of Closed eGovernments (refinement)* rule is a result of one of the improvements included in the *regulation of eGovernments* class, refined according to one of the improvements included in the *promotion of open standards* class. So, it will have a direct impact on the inefficiency that is a result of closed standards. But this impact is generally limited to software used in government communications only. Given the spill-over effect of eGovernments on software used outside of government communications, especially if the government chooses multi-purpose open standards, it may be also expected that this rule will affect users of such other software as well. So, closed standards will not be removed from the framework completely, but the prohibition of Closed eGovernments may materially minimize the negative impact that closed standards have on the freedoms.

The *guidelines for Supra-Open eGovernments* rule is a result of one of the improvements included in the *regulation of eGovernments* class. So, it is capable of affecting many different inefficiencies of the framework. The guidelines should in particular regulate how the government becomes involved in the development and distribution of Free Software. It means that the software resulting from such an involvement can be free of many inefficiencies that result from circumstances controlled by the government. Whether these inefficiencies will remain in the framework depends on the exact content of

the guidelines. Since this requires further empirical research, we are unable to determine the exact impact of the guidelines in this thesis. However, we may provide an example. Assume that the guidelines will prohibit governments to accept restrictive contracts on Free Software procured for eGovernments. In such a case the inefficiency that is a result of distributor contracts could be removed from the point of view users of such software, while it may still apply to software used outside government communications.

The *extended* copyleft *enforceability* is an example rule that could be created as a result of the implementation of improvements included in the *Free Software legislation* class. This rule addresses a specific inefficiency of the current framework, namely the *inter partes* nature of the framework. As a result of this rule, the *inter partes* nature would no longer constitute a burden for enforcing *copyleft* obligations. This means that we could remove the *inter partes* nature from the framework if the extended *copyleft* enforceability is implemented.

The *safe harbour against software-related patents* is a rule directed specifically at software-related patents. It does not remove these patents from the framework, but it limits their enforceability to the extent such an enforceability would constitute a threat to user freedoms. We remark that a complete removal of software-related patents – in such a way that they do not affect Free Software users at all – may be a result of other improvements included in the *restriction of software-related patents* class.

The *safe harbour against patents material to standards* is a rule similar to the safe harbour described above. It is a result of one of the improvements included in the *promotion of open standards* class. This rule does not remove the respective patents from the framework and it does not make it impossible to make a closed standard using patents. However, it prevents patent holders from enforcing such patents against users of Free Software.

The *limited enforceability of patents material to standards* works in a similar way. This rule also results from an implementation of the improvements included in the *promotion of open standards* class. It prevents patent holders from enforcing their patents in relation to programs that use standards covered with their patents provided that these standards have been offered and accepted in an eGovernment. This rule expresses a view that the government cannot support monopolies more than necessary. Thus, under this rule patent holders could not benefit from both the patent monopoly and the market power resulting from government demand (and the user demand induced by the government). Offering a technology to an eGovernment would require that patent rights are not enforceable towards the developers, distributors, and users of Free Software that makes use of that technology.

7.3.3 Conclusions on the provisional impact of our proposed framework

Here, we may conclude that the provisional impact of our proposed framework will be that at least seven new rules are created. Other new rules may follow depending on how the improvements are implemented exactly. The

new rules will enter into relations with the rules currently active in the framework. Six out of seven new rules that we identified above will limit the negative effect of the respective inefficiencies in such a way that the freedoms will be adequately protected. Only one of the seven new rules (extended *copyleft* enforceability) can be expected to remove its corresponding inefficiency from the framework (the *inter partes* nature of the licenses).

7.4 Further research

The proposal of an improved regulatory framework presented in this thesis includes improvements grouped in the following six sets: (1) proper organization of the communities, (2) regulation of eGovernments, (3) Free Software legislation, (4) restriction of software-related patents, (5) promotion of open standards, and (6) internationalization of the framework. In the above section we identified the provisional impact of our proposed framework, which consists of the creation of seven new rules, and of a removal of one inefficiency. Additional new rules may be created as well depending how the improvements are implemented exactly. So, the formulation of specific legal or non-legal regulators that would constitute the implementation of all improvements proposed in this thesis requires that the research is continued. In particular, there is a need for more empirical research that will provide more findings about circumstances for which the improvements are to be implemented. We indicate the following example areas for further research.

(1) **Free Software licenses and their impact under different jurisdictions.** The exact scope of obligations that follow from different licenses should be determined (such as the scope of *copyleft* clauses) so that the communities are able to account for the obligations (and the resulting incompatibilities) in all jurisdictions properly. This would also allow for a formulation of specific recommendations for amending licenses to become more jurisdiction-independent and effective.

(2) **Organization of the communities.** The ability of various communities to provide for proper maintenance of Free Software should be analysed and compared. A set of model organization structures should be identified. The best ways of adopting these structures in communities should be determined. In particular, it should be determined whether the organization of the communities should be regulated using the law, or whether the norms are sufficient.

(3) **Government role in the framework.** The ability of governments to become involved in the development, distribution, and use of Free Software should be studied in order to elaborate an adequate model of participation. At the same time governments should perform a survey of existing eGovernments in order to identify all closed

standards currently used and possibilities of providing for translation facilities so that all existing Closed eGovernments are transformed into Open eGovernments.

(4) **Particular national laws.** Particular national laws should be studied in order to identify all specific limitations and restrictions included therein. In other words, the research performed in this thesis should be repeated with regard to each national law separately. This would allow to propose specific Free Software legislation for each of the national laws.

(5) **Patent reform.** A thorough scrutiny of all existing patent laws and the practice of their application should be performed. Proposals for their reform should be elaborated. New ways of scrutinizing patent applications should be explored in order to increase the effectiveness of patent offices and to increase patent quality.

(6) **Standard setting.** Particular SSOs and procedures that they follow should be analysed and compared. All circumstances that prevent SSOs from adopting open standards should be identified and appropriate reforms of particular SSOs should be designed. Government awareness and involvement in standard setting should be increased.

(7) ***Lex mercatoria*.** Customs and trade practices of software communities should be analysed and particular rules that follow from them identified. Appropriate means for taking these rules into account when deciding disputes should be designed.

References

Alioth, *Virtual Stallman*, at: http://alioth.debian.org/projects/vrms/.

Apache, *How the ASF works*, at: http://www.Apache.org/foundation/how-it-works.html.

Attridge, Daniel J. M., *Challenging Claims! Patenting Computer Programs in Europe and the USA*, 1 Intellectual Property Quarterly 22 (2001).

autonomo.us, *Franklin Street Statement on Freedom and Network Services*, http://autonomo.us/2008/07/franklin-street-statement/.

Aviram, Amitai, *A Network Effects Analysis of Private Ordering*, (April 15, 2003), Berkeley Program in Law & Economics, Working Paper Series. Paper 80, http://repositories.cdlib.org/blewp/art8.

Barta, Janusz et al., Ustawa o prawie autorskim i prawach pokrewnych. Komentarz [Act on copyright and neighbouring rights. Commentary] (Dom Wydawniczy ABC 2001).

Barta, Janusz; Markiewicz, Ryszard, Oprogramowanie Open Source w świetle prawa. Między własnością a wolnością [Open source software in the light of law. Between property and freedom] (Zakamycze 2005).

Barta, Janusz; Markiewicz, Ryszard, Prawo autorskie [Copyright] (Wolters Kluwer 2008).

Barth, Andreas; Di Carlo, Adam; Hertzog, Raphaël; Schwarz, Christian; Jackson, Ian, *Debian Developer's Reference*, at: http://debian.org/doc/packaging-manuals/developers-reference/.

Bartlett, Andrew, *A year since Microsoft's appeal failed*, at: http://people.samba.org/people/abartlet/a-year-since-microsofts-appeal-failed.html.

Bender, David, *Trade Secret Implications of Open Source Licenses*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 129.

Benkler, Yochai, *An Unhurried View of Private Ordering in Information Transactions*, 53 Vand. L. Rev. 2063, (2000).

Benkler, Yochai, *Coase's Penguin, or, Linux and The Nature of the Firm*, 112 Yale Law Journal 369 (2002).

Business Software Alliance, *BSA Statement on Technology Standards*, February 2005, at: http://www.etsi.org/sos_interoperability/Background_papers/BSA_Statement_on_Technology_Standards.pdf.

Byfield, Bruce, *FSFE's Fiduciary License Agreement is no panacea*, at: http://www.linux.com/feature/60129.

Byrska, Małgorzata, *Prawne aspekty modyfikowania programu komputerowego [Legal aspects of modifying a computer program]*, 4 Kwartalnik Prawa Prywatnego [Private Law Quarterly] 693 (1996).

Chikofsky, E.J.; Cross II, J.H., *Reverse Engineering and Design Recovery: A Taxonomy in IEEE Software*, IEEE Computer Society: 13–17 (January 1990).

CNN, *IBM Pledges Free Access to Patents Involved in Implementing 150+ Software Standards*, at: http://money.cnn.com/news/newsfeeds/articles/marketwire/0276035.htm.

Coase, Ronald, *The Nature of the Firm*, Economica, vol. 4, no. 16, November 1937

Coleman, Gabriella E., The Social Construction of Freedom in Free and Open Source Software: Hackers, Ethics, and the Liberal Tradition (Dissertation, University of Chicago, 2005), at: http://www.healthhacker.org/biella/freesoftware.html.

Corbet, Jon, *How to Participate in the Linux Kernel Community*, at: http://ldn.linuxfoundation.org/book/how-participate-linux-community.

Cox, Alan, *Cathedrals, Bazaars and the Town Council*, at: http://slashdot.org/features/98/10/13/1423253.shtml.

Crowston, Kevin; Howison, James, *The social structure of free and open source software development*, Firstmonday, at: http://www.firstmonday.org/issues/issue10_2/crowston/index.html.
Cundiff, Victoria A., *Protecting Computer Software as a Trade Secret*, in: 507 Practising Law Institute, 18th Annual Institute on Computer Law 761 (1998).
Davidson, Stephen J.; Holloway, Gabriel, *Protecting Trade Secrets in an Open Source Environment*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 143.
Debian Free Software Guidelines, at: http://www.debian.org/social_contract.
debian-legal mailing list archives, at: http://lists.debian.org/debian-legal/.
Debian, *Debian New Maintainers*, at: http://debian.org/devel/join/newmaint.
DeKoenigsberg, Greg, *The Red Hat Patent Promise: Encouraging Innovation*, at: http://www.redhat.com/magazine/001nov04/features/patents/.
Demazerie, Didier; Horn, Francois; Jullien, Nicolas, *How free software developers work*, at: http://ssrn.com/abstract=1301572 (abstract).
Demazerie, Didier; Horn, Francois; Zune, Marc, *The Functioning of a Free Software Community: Entanglement of Three Regulation Modes – Control, Autonomous, and Distributed*, at: http://www.sciencestudies.fi/v20n2DemaziereHornZunePDF.
EC Communication, *The role of eGovernment for Europe's future* (COM(2003) 567, not published in OJ).
Economides, Nicholas, *The Economics of Networks*, 4 International Journal of Industrial Organization 673 (1996) at: http://www.stern.nyu.edu/networks/94-24.pdf.
Elliot, Margaret S.; Scacchi, Walt, *Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture*, in: Koch S. (ed.), Free/Open Source Software Development (Idea Publishing, 2004), book chapter at: http://www.ics.uci.edu/~wscacchi/Papers/New/Elliot-Scacchi-BookChapter.pdf.
EPO, *Guidelines for Examination in the European Patent Office (2009)*, at: http://www.epo.org/patents/law/legal-texts/guidelines.html.
EPO, *Pending referral to the Enlarged Board of Appeal (G 3/08)*, at: http://www.epo.org/topics/issues/computer-implemented-inventions/referral.html.
eWEEK, *Open-Source Insurance Provider Finds Patent Risks in Linux*, at: http://www.eweek.com/c/a/Linux-and-Open-Source/OpenSource-Insurance-Provider-Finds-Patent-Risks-in-Linux/.
FCKeditor, at: http://www.fckeditor.net.
Fedora Packaging Guidelines, at: http://fedoraproject.org/wiki/Packaging/Guidelines.
Fogel, Karl, *Producing Open Source Software. How to Run a Successful Free Software Project*, at: http://producingoss.com/.
Free Software Directory, at: http://directory.fsf.org/.
Free Software Foundation, *GNU General Public License Frequently Asked Questions*, at: http://www.fsf.org/licensing/licenses/gpl-faq.html.
Free Software Foundation, *Various Licenses and Comments About Them*, at: http://www.gnu.org/philosophy/license-list.html.
FreshMeat, at: http://freshmeat.net.
Fuchs, Bernadetta, Lex mercatoria w międzynarodowym obrocie handlowym [Lex mercatoria in international trade] (Zakamycze, 2000).
Garzarelli, Giampaolo, *Open Source Software and the Economics of Organization*, at: http://ideas.repec.org/p/wpa/wuwpio/0304003.html.
Ghosh, Rishab Aiyer; Glott, Ruediger; Robles, Gregorio; Schmitz, Patrice-Emmanuel, Guideline for Public Administrations on Partnering with Free Software Developers (European Commission, Enterprise DG, IDA/GPOSS, 2004), at: http://europa.eu.int/idabc/servlets/Doc?id=19295.
GPL-violations, at: http://gpl-violations.org.
Grierson, Kevin W., *Enforceability of „Clickwrap" or „Shrinkwrap" Agreements Common in Computer Software, Hardware, and Internet Transactions*, 106 American Law Reports 5th 309.
Groklaw, *RedHat is Asking for Prior Art*, at: http://www.groklaw.net/article.php?story=20090216150306923.

Gross, Alois Valerian, *What is Computer "Trade Secret" under State Law*, 53 American Law Reports 4th 1046.

Guibault, Lucie; van Daalen, Ot, Unraveling the Myth around Open Source Licenses. An Analysis from A Dutch and European Law Perspective (TMC Asser Press 2006).

Holbrook, Timothy R., *The paradoxical nature of U.S. patent scope*, in: Maciej Barczewski et. al. (eds.), When Worlds Collide: Intellectual Property, High Technology and the Law (Wolters Kluwer 2008), 65.

IBM, *IBM Pledges Free Access to Patents Involved in Implementing 150+ Software Standards*, at: http://www-03.ibm.com/press/us/en/pressrelease/21846.wss.

IBM, *The European Community Patent revisited. Discussion paper from IBM*, at: http://www.epip.eu/conferences/epip02/lectures/European%20Interoperabily%20Patent%201.1.pdf.

IDABC, *Linking up Europe: the Importance of Interoperability for eGovernment Services, Commission Staff Working Paper* (EC 2003), at: http://europa.eu.int/idabc/servlets/Doc?id=1675.

IDABC, *Translation of EUPL v.1.0 into the official languages of the European Union – Report on comments received by IDABC*, at: http://ec.europa.eu/idabc/servlets/Doc?id=29987.

Initiative for Software Choice, *New EU Public Procurement Directives – Maintaining Neutrality in Software Procurement*, September 2004, at: http://www.softwarechoice.org/download_files/ISC_LegalNote.pdf.

ISO/IEC Guide 2:2004 *Standardization and related activities – General vocabulary*.

ISO/IEC/ITU, *Common patent policy*, at: http://www.iso.org/patents.

Jensen, Chris; Scacchi, Walt, *Role Migration and Advancement Process in OSSD Projects: A Comparative Case Study*, at: http://opensource.mit.edu/papers/Jensen-Scacchi-ICSE-2007.pdf.

Krishnamurthy, Sandeep, *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*, Firstmonday, at: http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html.

Lakhani, K.R.; Wolf, B.; Bates J.; DiBona, C., *The Boston Consulting Group Hacker Survey*, at: http://www.osdn.com/bcg/bcg-0.73/img1.html.

Laurent, Andrew M. St., Understanding Open Source and Free Software Licensing (O'Reilly, 2004).

Lemley, Mark A., *Antitrust and the Internet Standardization Problem*, 28 Connecticut Law Review 1041.

Lemley, Mark A., *The Economics of Improvement in Intellectual Property Law*, 75 Texas Law Review 989 (1997).

Lessig, Lawrence, *The New Chicago School*, 27 The Journal of Legal Studies 661 (June 1998).

Levy, Steven, Hackers: Heroes of the Computer Revolution, (New York, Dell, 1985).

Liebowitz, S.J.; Margolis, Stephen E., *Network Externalities (Effects)*, at: http://www.utdallas.edu/~liebowit/palgrave/network.html.

Lindberg, Van, Intellectual Property and Open Source, (O'Reilly 2008).

Linux Foundation, *Linux Foundation Publishes Study on Linux Development*, http://linux-foundation.org/weblogs/press/2008/03/31/linux-foundation-publishes-study-on-linux-development-statistics-who-writes-linux-and-who-supports-it/.

Livna, at: http://livna.org.

Lloyd, Ian J., Information Technology Law, 308 (Butterworths, 2000, 3rd ed.).

Margolis, Stephen E.; Liebowitz, S.J., *Path dependence*, at: http://www.utdallas.edu/~liebowit/palgrave/palpd.html.

MeatBall, *RightToFork*, at: http://www.usemod.com/cgi-bin/mb.pl?RightToFork.

Meeker, Heather J., The Open Source Alternative Understanding Risks and Leveraging Opportunities (Wiley, 2008).

Ministerie van Economische Zaken, *Ontwerptekst instructie Comply or explain & commit*, at: http://www.ez.nl/Onderwerpen/Betrouwbare_telecom/Open_Standaarden_en_Open_Source_Software/Ontwerptekst_instructie_Comply_or_explain_commit.

Moglen, Eben, *Anarchism Triumphant: Free Software and the Death of Copyright*, at: http://emoglen.law.columbia.edu/my_pubs/anarchism.html.

Moglen, Eben, *Enforcing the GNU GPL*, at: http://www.gnu.org/philosophy/enforcing-gpl.html.

Moglen, Eben, *The GPL Is a License, not a Contract*, at: http://lwn.net/Articles/61292/.
Mueller, Janice M., *Patent Misuse Through the Capture of Industry Standards*, 17 Berkeley Technology Law Journal 623 (2002).
Murphy, Gary, *The Linux Kernel, Blueprints for World Domination*, at: http://kernelbook.sourceforge.net/pkbook.html.
Nichols, Kenneth, Inventing software: the rise of "computer-related" patents (Quorum Books, 1998).
Nimmer, Raymond T., *Legal Issues in Open Source and Free Software Distribution*, Open Source Software Spring 2006 Critical Issues in Today's Corporate Environment, PLI Handbook no. G-861, 7.
NOiV (OSSOS), *The acquisition of (open-source) software, A guide for ICT buyers in the public and semi-public sectors*, at: http://ososs.nl/files/acquisition_of_open-source_software_-_text.pdf.
Novell, *Patent Policy*, at: http://www.novell.com/company/policies/patent/.
Ogden, Christopher L., *Patentability of Algorithms After State Street Bank: The Death of the Physicality Requirement*, No. 10 Vol. 82 Journal of Patent and Trademark Office Society 721, 724 *et seq* (2000).
Open Forum Europe, *How Open Can Europe Get*, at: http://www.openforumeurope.org/index.php?option=com_docman&task=doc_download&gid=89&Itemid=102.
Open Invention Network, at: http://www.openinventionnetwork.com/.
Open Source Initiative, *Open Source Definition*, at: http://opensource.org/docs/definition.php.
Open Source Initiative, *The Approved Licenses*, at: http://opensource.org/licenses/.
OpenOffice, *Sun Contributorship Agreement*, at: http://contributing.openoffice.org/programming.html#sca.pdf.
OpenSuSE, *OpenSuSE License*, at: http://en.opensuse.org/OpenSUSE_License.
Patently O, *In re Bilski*, at: http://www.patentlyo.com/patent/2008/10/in-re-bilski.html.
Peer-to-Patent, at: http://www.peertopatent.org/.
Perens, Bruce, *Open Standards Principles and Practice*, at: http://perens.com/OpenStandards/Definition.html.
Polański, Przemysław P., Customary Law of the Internet (TMC Asser Press 2007).
Prince, Brian, *Sun Asserts MySQL Will Remain Open Source*, at: http://www.eweek.com/c/a/Database/Sun-Asserts-MySQL-to-Remain-Open-Source/.
Raimondo, F. O., General Principles of Law in the Decisions of International Criminal Court and Tribunals (2007, Ph.d. thesis at University of Amsterdam)
Ramos, Carey R.; Berlin, David S., *Three Ways to Protect Computer Software*, 16 No. 1 Computer Lawyer 16 (1999)
Raymond, Eric S., *A Brief History of Hackerdom*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/hacker-history/index.html.
Raymond, Eric S., *Homesteading the Noosphere*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading/.
Raymond, Eric S., *How to Become a Hacker*, at: http://www.catb.org/~esr/writings/faqs/hacker-howto.html).
Raymond, Eric S., *The Cathedral and the Bazaar*, at: www.catb.org/~esr/writings/cathedral-bazaar/.
Raymond, Eric S., *The Magic Cauldron*, at: http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/.
Reichman, J.H.; Franklin, Jonathan A., *Privately Legislated Intellectual Property Rights: Reconciling Freedom of Contract with Public Good Uses of Information*, 147 U. Pa. L. Rev. 875 (1999).
RFC1392, at: http://rfc.net/rfc1392.html.
Rosen, Lawrence, *Bad Facts Make Good Law: The* Jacobsen *Case and Open Source*, at: http://www.rosenlaw.com/BadFactsMakeGoodLaw.pdf.
Rullani, Francesco, *Dragging developers towards the core*, at: ftp://ftp.unibocconi.it/pub/RePEc/cri/papers/WP190Rullani.pdf.
Sachs, Stephen Edward, *From St. Ives to Cyberspace: The Modern Distortion of the Medieval 'Law Merchant'* 21 American University International Law Review 685 (2006), also available at: http://ssrn.com/id=830265.

Saltzer, J.H; Reed, D.P.; Clark, D.D., *End-to-End Argument in System Design*, 2 ACM Transactions in Computer Systems 277 (1984), at: http://web.mit.edu/Saltzer/www/publications/endtoend/endtoend.pdf).

Samuelson, Pamela, *Reverse Engineering Under Siege*, 10 Communications of the ACM 15 (2002).

Scherer, F.M., *Microsoft and IBM in Europe*, 2090 Antitrust & Trade Regulation Report 65, 66 (2003).

Schiuma, Daniele, *TRIPS and Exclusion of Software "as Such" from Patentability*, Vol. 31, No.1, IIC International Review of Industrial Property and Copyright Law 36 (2000).

Scott, Brendan, *BSD – The Dark Horse of Open Source*, at: http://opensourcelaw.biz/publications/papers/BScott_BSD_The_Dark_Horse_of_Open_Source_070112lowres.pdf.

Siewicz, Krzysztof *Scope of copyleft clause under Polish law*, in: Barta J. (ed.), Zagadnienia Prawa Autorskiego [*Copyright Law Issues*], ZNUJ PWiOWI [Jagiellonian University Intellectual Property Journal], Vol. 93 p. 235, (Zakamycze 2006), English version at: http://ksiewicz.net/files/siewicz_copyleft_scope.pdf.

Skulikaris, Yannis, *Software-Related Inventions and Business-Related Inventions; A review of practice and case law in U.S. and Europe*, Patent World, February 2001, 26.

Slashdot, *Hans Reiser Speaks Freely About Free Software Development*, at: http://developers.slashdot.org/article.pl?sid=03/06/18/1516239&tid=156&tid=11.

Software Freedom Law Center, *A Legal Issues Primer for Open Source and Free Software Projects*, at: http://www.softwarefreedom.org/resources/2008/foss-primer.html.

Software Freedom Law Center, at: http://softwarefreedom.org.

Software Freedom Law Center, *Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers,* at: http://softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html.

Solum, Lawrence B.; Chung, Minn, *The Layers Principle: Internet Architecture and the Law*, 79 Notre Dame Law Review 815 (2004).

SourceForge, at: http://sourceforge.net.

Stallman, Richard M., *Free Software Definition*, at: http://www.gnu.org/philosophy/free-sw.html.

Stallman, Richard M., *Why "Free Software" is better than "Open Source"*, at: http://www.fsf.org/licensing/essays/free-software-for-freedom.html.

Sundararajan, Arun, *Network Effects*, at: http://oz.stern.nyu.edu/io/network.html.

The Fedora Extras license audit, at: http://lwn.net/Articles/218977/.

The Register, *EC acts on patent ambushes*, at: http://www.theregister.co.uk/2005/12/14/patent_ambush/.

Toren, Peter, *Software and Business Methods are Patentable in the U.S. (Get over it)*, Patent World, September 2000, 8).

Tripathi, R C *et al.*, *Patenting of Computer Software: Status and Approach*, Vol. 7 Journal of Intellectual Property Rights 128 (2002).

Tsilas, Nicos L., *The Threat to Innovation, Interoperability, and Government Procurement Options From Recently Proposed Definitions of "Open Standards"*, Special Issue Global Flow of Information, Autumn 2005 (at: http://www.ijclp.org/10_2005/pdf/ijclp_08_10_2005.pdf).

Turney, James, *Defining the Limits of the EU Essential Facilities Doctrine on Intellectual Property Rights: The Primacy of Securing Optimal Innovation*, 2 Northwestern Journal of Technology and Intellectual Property 179 (2005).

Ubuntu Daily, *Proprietary drivers in Feisty: not by default but easy to activate*, at: http://ubuntudaily.com/2007/03/06/proprietary-drivers-in-feisty-not-by-default-but-easy-to-activate/.

Ubuntu, *Licensing*, at: http://www.ubuntu.com/community/ubuntustory/licensing.

Ubuntu, *What is Ubuntu?*, at: http://www.ubuntu.com/products/whatisubuntu.

Vardakas, Evangelos, *The role of government in standards setting: a European View*, 5, in: Vademecum on European Standardization, Part II, Chapter 1, at: http://europa.eu.int/comm/enterprise/standards_policy/vademecum/doc/standards_setting_governance_ev.pdf).

Vetter, Greg R., *„Infectious" Open Source Software: Spreading Incentives or Promoting Resistance?*, 36 Rutgers Law Journal 53 (2004).
Waglowski, Piotr, *Sąd Okręgowy oddalił apelację ZUS w sprawie protokołu KSI-MAIL. Koniec*, *[Circuit court rejected the appeal of ZUS in the KSI-MAIL case. The end.]*, at: http://prawo.vagla.pl/node/7222.
Weber, Steven, The Success of Open Source (Harvard University Press, 2004).
Welte, Harald, *Some more thoughts on the results of GPL enforcement*, at: http://gnumonks.org/~laforge/weblog/2006/10/30/#20061030-gpl-devices.
Wiewiórowski, Wojciech, *Komunikat dotyczący Edytora Aktów Prawnych*, at: http://bip.mswia.gov.pl/portal/bip/21/17881/.
Wikipedia, *Android (mobile device platform)*, at: http://en.wikipedia.org/wiki/Android_(mobile_device_platform).
Wikipedia, *BDI software agent*, at: http://en.wikipedia.org/wiki/BDI_software_agent.
Wikipedia, *CentOS*, at: http://en.wikipedia.org/wiki/Centos.
Wikipedia, *Clean room design*, at: http://en.wikipedia.org/wiki/Clean_room_design.
Wikipedia, *Cloud computing*, at:http://en.wikipedia.org/wiki/Cloud_computing.
Wikipedia, *Corporate veil*, at:http://en.wikipedia.org/wiki/Corporate_veil.
Wikipedia, *Decompilation*, at: http://en.wikipedia.org/wiki/Decompilation.
Wikipedia, *Doctrine of equivalents*, at:http://en.wikipedia.org/wiki/Doctrine_of_Equivalents.
Wikipedia, *eGovernment*, at: http://en.wikipedia.org/wiki/eGovernment.
Wikipedia, *Essential Facilities Doctrine*, at: http://en.wikipedia.org/wiki/Essential_facilities.
Wikipedia, *Fork (software development)*, at: http://en.wikipedia.org/wiki/Fork_(software_development).
Wikipedia, *Juraj Janosik*, at: http://en.wikipedia.org/wiki/Juraj_J%C3%A1no%C5%A1%C3%ADk.
Wikipedia, *Law Merchant*, at: http://en.wikipedia.org/wiki/Lex_mercatoria#The_medieval_Law_Merchant.
Wikipedia, *Metcalfe's law*, at: http://en.wikipedia.org/wiki/Metcalfe%27s_law.
Wikipedia, *Network effects*, at: http://en.wikipedia.org/wiki/Network_effects.
Wikipedia, *OSI Model*, at: http://en.wikipedia.org/wiki/OSI_model.
Wikipedia, *Patent ambush*, at: http://en.wikipedia.org/wiki/Patent_ambush.
Wikipedia, *Program library*, at:http://en.wikipedia.org/wiki/Program_library.
Wikipedia, *Programming language*, at: http://en.wikipedia.org/wiki/Programming_language.
Wikipedia, *Reverse engineering*, at: http://en.wikipedia.org/wiki/Reverse_engineering.
Wikipedia, *Self-modifying code*, at: http://en.wikipedia.org/wiki/Self-modifying_code.
Wikipedia, *Tivoization,* at:http://en.wikipedia.org/wiki/Tivoization.
Wikipedia, *Transaction costs*, at:http://en.wikipedia.org/wiki/Transaction_costs.
Williamson, Oliver E., *The Economics of Governance*, at: http://www.aeaweb.org/annual_mtg_papers/2005/0107_1645_0101.pdf.
Williamson, Oliver E., *The Theory of the Firm as Governance Structure: From Choice to Contract*, 3 Journal of Economic Perspectives 171 (Summer 2002).
Williamson, Oliver E., *Why Law, Economics, and Organization?*, at: http://papers.ssrn.com/paper.taf?abstract_id=255624.
Wine, at: http://winehq.org.
WIPO (Standing Committee on the Law of Patents), *Report on the International Patent System*, (3 February 2009, SCP/12/3 Rev. 2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_12/scp_12_3.pdf.
WIPO (Standing Committee on the Law of Patents), *Standard and Patents*, (18 February 2009, SCP/13/2), at: http://www.wipo.int/edocs/mdocs/scp/en/scp_13/scp_13_2.pdf.
Worthington, David, *Experts: Microsoft's FAT licensing terms might violate GPL*, at: http://www.sdtimes.com/link/33327.
Worthington, David, *TomTom can license FAT without violating GPL*, at: http://www.sdtimes.com/blog/1364.
ZUS, at: http://zus.pl.
ZUS, *Wymagania dla oprogramowania interfejsowego, [Requirements for interface software]*, at: http://www.zus.pl/bip/default.asp?id=180.

Summary

This thesis concerns the protection of users of computer programs known as Free Software. It is based on the premise that a worldwide regulatory framework should be adopted to protect the users. In this thesis, a model of the current framework is reconstructed and its operation analysed in the world of software communities and eGovernments. The thesis also proposes an improved framework, intended to protect the freedoms of users more adequately than in the current framework.

In Chapter 1 I provide an introduction to the analysis of the current framework, by presenting the scene of play together with the actors and the audience of that scene. This leads to one of the most important findings, viz. the control of the working of the programs (and essentially the protection of the user freedoms) require(s) that two conditions are met. The conditions are: (1) access to source codes of the programs, as well as (2) access to the specifications of standards used by the programs to interoperate with other programs. Then, by looking at the software communities and eGovernments in relation to user freedoms as defined by Stallman, I formulate a Research Goal and the Problem Statement. Thereafter, I formulate three Research Questions to be answered by analysing the current regulatory framework and by proposing an improved framework. Finally, I present the research methodology to be used in the analysis.

In Chapter 2 I provide definitions and basic notions that are used throughout the analysis. These are (1) Free Software, (2) Open Source Software, (3) open standards, (4) software communities, and (5) eGovernments. In particular, I explain what are the subject matter, the requirements, and the addressees of the Free Software Definition. It leads to the finding that Free Software refers to computer programs available together with the rights to use, develop, and distribute them. In other words, the Free Software Definition requires that users are allowed to exercise all activities covered by copyright.

In Chapter 3 I reconstruct a model of the current regulatory framework of Free Software. I include in the framework rules that regulate (1) the access to software and standards, and (2) the relations between the rules. As a result, the model reconstructed in Chapter 3 consists of rules that enable users to exercise their freedoms, as well as the rules that limit or restrict users in exercising them (the latter jointly referred to as the "inefficiencies"). The inefficiencies are: (1) license proliferation and incompatibility, (2) license revocability, (3) *inter partes* nature of licenses, (4) software-related patents, (5) contracts with distributors, (6) liability rules, (7) non-legal regulators of software, (8) closed standards, and (9) regulatory environment. I complete Chapter 3 by a conclusion that many of the inefficiencies are not sufficiently

addressed by the model. It follows that the freedoms are not sufficiently protected under the current framework.

In Chapter 4 I analyse how software communities affect the protection of user freedoms under the current framework. This is done by identifying relations between the communities and the model of the framework. The analysis of software communities leads to a conclusion that properly organized communities are able to gather and manage resources necessary to minimize many limitations and restrictions of the current framework materially. However, the inefficiencies as such remain. The inefficiencies that are the least affected by the communities are: (1) software-related patents, (2) closed standards, and (3) the regulatory environment.

In Chapter 5 I analyse how eGovernments affect the protection of user freedoms under the current framework. I perform this analysis separately for Closed eGovernments (*i.e.*, eGovernments based on closed standards) and for Open eGovernments (*i.e.*, eGovernments based on open standards). It leads to a conclusion that the limitations and restrictions of the current framework continue to exist in the world of eGovernments. However, eGovernments are able to affect how the inefficiencies of the current framework impact user freedoms. They may do so in both ways. Generally, Closed eGovernments lead to further restriction of user freedoms, despite the protection inherent in the current framework, and despite any positive effect that the communities might have on the freedoms. Conversely, Open eGovernments are mostly neutral towards the limitations and restrictions, although they affect some of them to a benefit of the freedoms.

In Chapter 6 I propose an improved regulatory framework of Free Software. The proposal originates from the findings and conclusions presented in previous chapters. It is started by a recapitulation of all identified inefficiencies of the current framework and a discussion about possible improvements of each of them. Then, a selection of the most appropriate improvements is performed. The improvements are grouped under the following headings: (1) proper organization of communities, (2) guidelines for eGovernments, (3) Free Software legislation, (4) restriction of software-related patents, (5) promotion of open standards, and (6) internationalization of the framework. I then add the improvements to the model of the current framework, and present the relations between the improvements and other rules in the model. As a result, I construct the proposal of an improved regulatory framework.

In Chapter 7 I finalize the research. I recall the answers to the three Research Questions, as well as the answers to the Problem Statement (each of them is given after a relevant part of the analysis in previous chapters). The chapter ends with an identification of issues for further research.

Towards_an_Improved_def.indd 230 23-02-2010 10:49:40

Samenvatting

Op weg naar een betere rechtsbescherming van Free Software

Dit proefschrift gaat over de bescherming van gebruikers van de zogeheten "vrije software" (Free Software). Het gaat uit van de premisse dat er, om die gebruikers te beschermen, een wereldomvattend regulerend kader zou moeten worden aangenomen. In dit proefschrift wordt een model van het bestaande kader gereconstrueerd en de werking daarvan geanalyseerd in de omgeving van "software communities" en "eGovernments". Er wordt daarnaast een verbeterd kader voorgesteld, dat tot doel heeft de vrijheden van de gebruikers een meer adequate bescherming te bieden dan het bestaande kader doet.

In Hoofdstuk 1 geef ik een inleiding op de analyse van het huidige kader aan de hand van een schets van het speelveld, van de actoren en van de toeschouwers van dat speelveld. Dit leidt tot een van de belangrijkste constateringen, namelijk de controle over de werking van de programmatuur (en in essentie de bescherming van de vrijheden voor de gebruiker) vereist (vereisen) dat wordt voldaan aan twee voorwaarden. Deze voorwaarden zijn:
- toegang tot de broncode van de programmatuur; en
- toegang tot de specificaties van de standaarden die de programmatuur gebruikt vanwege de interoperabiliteit met andere programmatuur.

Na aandacht te hebben besteed aan de "software communities" en "eGovernments" in relatie tot de vrijheden voor de gebruiker zoals gedefinieerd door Stallman, worden het onderzoeksdoel en de probleemstelling geformuleerd. Hierna formuleer ik drie onderzoeksvragen die ik wil beantwoorden door het analyseren van het bestaande regulerende kader en door een verbeterd kader voor te stellen. Ten slotte geef ik aan met behulp van welke onderzoeksmethodologie ik mijn analyse uitvoer.

In Hoofdstuk 2 behandel ik definities en begrippen zoals ik die ik in mijn analyse gebruik. Dit zijn:
- vrije software (Free Software);
- Open Source Software;
- Open standaarden;
- Software communities; en
- eGovernments.

Ik leg in het bijzonder uit wat de essentie is van de vrije software definitie (Free Software Definition), en om welke vereisten en geadresseerden het hierbij gaat. Dit leidt tot de conclusie dat vrije software (Free Software) computerprogrammatuur betreft die beschikbaar is tezamen met het recht om deze te gebruiken, te ontwikkelen en te verspreiden. Met andere woorden, de vrije software definitie (Free Software Definition) vereist dat gebruikers alle auteursrechtelijk relevante handelingen mogen verrichten.

In Hoofdstuk 3 reconstrueer ik een model van het huidige regulerende kader voor vrije software (Free Software). Ik behandel in het kader tevens de regels over:
– de toegang tot software en standaarden; en
– de relaties tussen de regels.
Het resultaat van deze reconstructie is een model dat bestaat zowel uit regels die gebruikers in staat stellen om hun vrijheden uit te oefenen, als uit regels die gebruikers uitsluiten of beperken in de uitoefening van de vrijheden (de laatste regels worden gezamenlijk aangeduid als de "ondoelmatigheden"). De ondoelmatigheden zijn:
– proliferatie van de licentie en incompatibiliteit;
– herroepbaarheid van de licentie;
– het *inter partes* karakter van de licenties;
– softwaregerelateerde octrooien;
– contracten met distributeurs;
– aansprakelijkheidsbepalingen;
– niet-juridische regelgeving van software;
– gesloten standaarden; en
– regelgevende omgevingen.
Ik sluit Hoofdstuk 3 af met een conclusie dat veel van de ondoelmatigheden niet voldoende worden bestreken door het model. Hieruit volgt dat de vrijheden met het bestaande kader niet voldoende worden beschermd.

In Hoofdstuk 4 analyseer ik hoe de software communities onder het bestaande kader de bescherming van de vrijheden van de gebruiker beïnvloeden. Zij doen dit door het stellen van relaties tussen de communities en het model van het kader. De analyse van software communities leidt tot een conclusie dat goed georganiseerde communities in staat zijn om bronnen te vergaren en te organiseren, die noodzakelijk zijn om veel uitsluitingen en beperkingen van het bestaande kader aanzienlijk te minimaliseren. Echter, de ondoelmatigheden als zodanig blijven bestaan. De ondoelmatigheden die het minst worden geraakt door de communities zijn:
– softwaregerelateerde octrooien;
– gesloten standaarden; en
– regelgevende omgevingen.

In Hoofdstuk 5 analyseer ik hoe eGovenments met het bestaande kader de bescherming van de vrijheden van de gebruiker beïnvloeden. Ik voer de analyse uit voor zowel de "Gesloten eGovernments" (d.w.z. eGovernments gebaseerd op gesloten standaarden) als voor "Open eGovernments" (d.w.z. eGovernments gebaseerd op open standaarden). Dit leidt tot de conclusie dat de uitsluitingen en beperkingen van het bestaande kader in de wereld van de eGovernments blijven bestaan. Echter, eGovernments zijn in staat om invloed uit te oefenen op de wijze waarop het bestaande kader inwerkt op de vrijheden van de gebruiker. Beide kunnen dat bewerkstelligen. Over het algemeen leiden Gesloten eGovernments tot een verdere beperking van de

vrijheden van de gebruiker, ondanks de bescherming die eigen is aan het bestaande kader, en ondanks positieve effecten die communities zouden kunnen hebben op de vrijheden. Daarentegen staan Open eGovernments doorgaans neutraal tegenover de uitsluitingen en beperkingen, ofschoon zij sommige daarvan beïnvloeden ten faveure van de vrijheden.

In Hoofdstuk 6 stel ik een verbeterd regulerend kader voor vrije software (Free Software) voor. Dit voorstel volgt uit de bevindingen en conclusies zoals weergegeven in de voorafgaande hoofdstukken. Het begint met het recapituleren van alle gestelde ondoelmatigheden van het bestaande kader en een discussie over mogelijke verbeteringen van elk daarvan. Vervolgens wordt een selectie gemaakt van de meest aangewezen verbeteringen. Deze worden gegroepeerd onder de volgende noemers:
– goede organisatie van communities;
– richtlijnen voor eGovernments;
– vrije software (Free Software) regelgeving;
– beperking van softwaregerelateerde octooien;
– promotie van open standaarden; en
– internationalisering van het kader.

Ik voeg daarna de verbeteringen toe aan het model van het bestaande kader en geef de relaties aan tussen de verbeteringen en de overige regels van het model. Het resultaat dat ik dan ontwerp is mijn voorstel voor een verbeterd regulerend kader.

In Hoofdstuk 7 sluit ik het onderzoek af met zowel de antwoorden op de drie onderzoeksvragen als de antwoorden op de probleemstelling (waarvan bij elk gegeven antwoord wordt ingegaan op betreffend relevant deel van de analyse zoals deze in de voorafgaande hoofdstukken is behandeld). Het hoofdstuk eindigt met het aangeven van punten die verder onderzoek rechtvaardigen.

Curriculum vitae

Krzysztof Siewicz was born in Warsaw, Poland, in 1979. He graduated from the Tadeusz Czacki high school in Warsaw, in 1998. From 1998 to 2003 he studied law at the Faculty of Law and Administration of the University of Warsaw. He completed his studies with a *summa cum laude* master of law diploma. Immediately after graduation he left Poland to attend a one-year course in the international business law at the Central European University in Budapest, Hungary. He finished the course with honours and obtained an LLM diploma in 2004.

In 2005 he joined the law firm Grynhoff Woźny & Partners, a renowned partnership of Polish lawyers specializing in telecommunications, media, and transportation law (currently, Grynhoff Woźny Maliński, associated with Bird&Bird). At the same time he started working on this doctoral thesis, after being admitted as a doctoral candidate by the Leiden University, eLaw@Leiden. His 6-month sabbatical in Leiden during the 2005/2006 academic year was funded by the Netherlands organization for international cooperation in higher education (Nuffic), as a part of the Huygens Scholarship Programme. The remaining part of his doctoral research was self-funded.

Krzysztof is also the legal lead of Creative Commons Poland, and runs a weblog about legal issues of Free Software at http://ksiewicz.net (in Polish, including English versions of some articles).

Towards_an_Improved_def.indd 234 23-02-2010 10:49:41

SIKS Dissertation Series

1998

1 Johan van den Akker (CWI[321]) *DEGAS – An Active, Temporal Database of Autonomous Objects*
2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*
3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*
4 Dennis Breuker (UM) *Memory versus Search in Games*
5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*
2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*
3 Don Beal (UM) *The Nature of Minimax Search*
4 Jacques Penders (UM) *The Practical Art of Moving Physical Objects*
5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*
6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*
7 David Spelt (UT) *Verification Support for Object Database Design*
8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

2000

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*
2 Koen Holtman (TU/e) *Prototyping of CMS Storage Management*
3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*
4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*
5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*
6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*
7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*
8 Veerle Coupé (EUR) *Sensitivity Analysis of Decision-Theoretic Networks*
9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*
10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*
11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*
2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*
3 Maarten van Someren (UvA) *Learning as Problem Solving*

---

321 Abbreviations: SIKS – Dutch Research School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; KUB – Katholieke Universiteit Brabant, Tilburg; KUN – Katholieke Universiteit Nijmegen; OU – Open Universiteit; RUL – Rijksuniversiteit Leiden; RUN – Radboud Universiteit Nijmegen; TUD – Technische Universiteit Delft; TU/e – Technische Universiteit Eindhoven; UL – Universiteit Leiden; UM – Universiteit Maastricht; UT – Universiteit Twente, Enschede; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; UvT – Universiteit van Tilburg; VU – Vrije Universiteit, Amsterdam.

4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*
6 Martijn van Welie (VU) *Task-Based User Interface Design*
7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*
8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*
9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*
10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*
11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*
2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*
3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*
4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*
5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments Inhabited by Privacy-Concerned Agents*
6 Laurens Mommers (UL) *Applied Legal Epistemology; Building a Knowledge-based Ontology of the Legal Domain*
7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*
8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*
9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*
10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*
11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*
12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*
13 Hongjing Wu (TU/e) *A Reference Architecture for Adaptive Hypermedia Applications*
14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*
15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*
16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*
17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*
2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*
3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*
4 Milan Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*
5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law -- A Modelling Approach*
6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*
7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*
8 Yong-Ping Ran (UM) *Repair-Based Scheduling*
9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*
10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*
12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*
13 Jeroen Donkers (UM) *Nosce Hostem -- Searching with Opponent Models*
14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*
15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*
16 Menzo Windhouwer (CWI) *Feature Grammar Systems – Incremental Maintenance of Indexes to Digital Media Warehouse*
17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*
18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*
2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*
3 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*
4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*
5 Viara Popova (EUR) *Knowledge Discovery and Monotonicity*
6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*
7 Elise Boltjes (UM) *Voorbeeldig Onderwijs; Voorbeeldgestuurd Onderwijs, een Opstap naar Abstract Denken, vooral voor Meisjes*
8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale Politiële Gegevensuitwisseling en Digitale Expertise*
9 Martin Caminada (VU) *For the Sake of the Argument; Explorations into Argument-based Reasoning*
10 Suzanne Kabel (UvA) *Knowledge-rich Indexing of Learning-objects*
11 Michel Klein (VU) *Change Management for Distributed Ontologies*
12 The Duy Bui (UT) *Creating Emotions and Facial Expressions for Embodied Agents*
13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*
14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*
15 Arno Knobbe (UU) *Multi-Relational Data Mining*
16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*
17 Mark Winands (UM) *Informed Search in Complex Games*
18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*
19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*
20 Madelon Evers Nyenrode *Learning from Design: facilitating multidisciplinary designteams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*
2 Erik van der Werf (UM) *AI techniques for the game of Go*
3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*
4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*
5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*
6 Pieter Spronck (UM) *Adaptive Game AI*
7 Flavius Frasincar (TU/e) *Hypermedia Presentation Generation for Semantic Web Information Systems*
8 Richard Vdovjak (TU/e) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
9 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*
10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering – A Decentralized Approach to Search*

12 Csaba Boer (EUR) *Distributed Simulation in Industry*
13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*
15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*
16 Joris Graaumans (UU) *Usability of XML Query Languages*
17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*
18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*
19 Michel van Dartel (UM) *Situated Representation*
20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*
21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TU/e) *Foundations of B2B Electronic Contracting*
2 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*
3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*
4 Marta Sabou (VU) *Building Web Service Ontologies*
5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*
6 Ziv Baida (VU) *Software-aided Service Bundling – Intelligent Methods & Tools for Graphical Service Modeling*
7 Marko Smiljanic (UT) *XML schema matching -- balancing efficiency and effectiveness by means of clustering*
8 Eelco Herder (UT) *Forward, Back and Home Again – Analyzing User Behavior on the Web*
9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*
10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*
11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*
12 Bert Bongers (VU) *Interactivation – Towards an e-cology of people, our technological environment, and the arts*
13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*
14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign – towards a Theory of Requirements Change*
15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*
16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*
17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*
18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*
19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*
20 Marina Velikova (UvT) *Monotone models for prediction in data mining*
21 Bas van Gils (RUN) *Aptness on the Web*
22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*
23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*
24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*
25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*
26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*
28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures}*
2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*
3 Peter Mika (VU) *Social Networks and the Semantic Web*

4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*
5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*
6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*
7 Natasa Jovanovic' (UT) *To Whom It May Concern – Addressee Identification in Face-to-Face Meetings*
8 Mark Hoogendoorn (VU) *Modeling of Change in Multi-Agent Organizations*
9 David Mobach (VU) *Agent-Based Mediated Service Negotiation*
10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*
11 Natalia Stash (TU/e) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*
12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*
13 Rutger Rienks (UT) *Meetings in Smart Environments; Implications of Progressing Technology*
14 Niek Bergboer (UM) *Context-Based Image Analysis*
15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*
16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*
17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*
18 Bart Orriens (UvT) *On the development and management of adaptive business collaborations*
19 David Levy (UM) *Intimate relationships with artificial partners*
20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*
21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*
22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*
23 Peter Barna (TU/e) *Specification of Application Logic in Web Information Systems*
24 Georgina Ramìrez Camps (CWI) *Structural Features in XML Retrieval*
25 Joost Schalken (VU) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbàn (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*
2 Alexei Sharpanskykh (VU) *On Computer-Aided Methods for Modeling and Analysis of Organizations*
3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*
4 Ander de Keijzer (UT) *Management of Uncertain Data – towards unattended integration*
5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*
6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*
7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*
8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*
9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*
10 Wauter Bosma (UT) *Discourse oriented Summarization*
11 Vera Kartseva (VU) *Designing Controls for Network Organizations: a Value-Based Approach*
12 Jozsef Farkas (RUN) *A Semiotically oriented Cognitive Model of Knowlegde Representation*
13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*
14 Arthur van Bunningen (UT) *Context-Aware Querying; Better Answers with Less Effort*
15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*
16 Henriette van Vugt (VU) *Embodied Agents from a User's Perspective*
17 Martin Op't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*

18 Guido de Croon (UM) *Adaptive Active Vision*
19 Henning Rode (UT) *From document to entity retrieval: improving precision and performance of focused text search*
20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met een overheid op de administratieve lasten van bedrijven*
21 Krisztian Balog (UvA) *People search in the enterprise*
22 Henk Koning (UU) *Communication of IT-architecture*
23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*
24 Zharko Aleksovski (VU) *Using background knowledge in ontology matching*
25 Geert Jonker (UU) *Efficient and Equitable exchange in air traffic management plan repair using spender-signed currency*
26 Marijn Huijbregts (UT) *Segmentation, diarization and speech transcription: surprise data unraveled*
27 Hubert Vogten (OU) *Design and implementation strategies for IMS learning design*
28 Ildiko Flesh (RUN) *On the use of independence relations in Bayesian networks*
29 Dennis Reidsma (UT) *Annotations and subjective machines- Of annotators, embodied agents, users, and other humans*
30 Wouter van Atteveldt (VU) *Semantic network analysis: techniques for extracting, representing and querying media content*
31 Loes Braun (UM) *Pro-active medical information retrieval*
32 Trung B. Hui (UT) *Toward affective dialogue management using partially observable markov decision processes*
33 Frank Terpstra (UvA) *Scientific workflow design; theoretical and practical issues*
34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*
35 Benjamin Torben-Nielsen (UvT) *Dendritic morphology: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*
2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*
3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*
4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*
5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks – Based on Knowledge, Cognition, and Quality*
6 Muhammad Subianto (UU) *Understanding Classification*
7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*
8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*
10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*
11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*
12 Peter Massuthe TU/e, Humboldt-Universtät zu Berlin *Operating Guidelines for Services*
13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*
14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*
15 Rinke Hoekstra (UvA) *Ontology Representation – Design Patterns and Ontologies that Make Sense*
16 Fritz Reul (UvT) *New Architectures in Computer Chess*
17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*
18 Fabian Groffen (CWI) *Armada, An Evolving Database System*
19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*
20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*
21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*

23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*
24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*
25 Alex van Ballegooij (CWI) *'RAM: Array Database Management through Relational Mapping'*
26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*
30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*
31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*
32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*
33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*
35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*
36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*
37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*
38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*
39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) *Service Substitution -- A Behavioral Approach Based on Petri Nets*
40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*
41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*
42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*
43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*
45 Jilles Vreeken (UU) *Making Pattern Mining Useful*
46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*
2 Ingo Wassink (UT) *Work flows in Life Science*
3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*
4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*
5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*
6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*
7 Wim Fikkert (UT) *A Gesture interaction at a Distance*
8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*
9 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*
10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*
11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*
12 Susan van den Braak (UU) *Sensemaking software for crime analysis*
13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*
14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*
15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*
16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*
17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*

Meijers Ph.D. list

In de boekenreeks van de Graduate School of Legal Studies van de Faculteit der Rechtsgeleerdheid, Universiteit Leiden, zijn in 2009 en 2010 verschenen:

MI-156 N.M. Dane, *Overheidsaansprakelijkheid voor schade bij legitiem strafvorderlijk handelen*, (diss. Leiden), Tilburg: Celsus juridische uitgeverij 2009, ISBN 978 90 8863 034 7

MI-157 G.J.M. Verburg, *Vaststelling van smartengeld,* (diss. Leiden) Deventer: Kluwer 2009

MI-158 J. Huang, *Aviation Safety and ICAO*, (diss. Leiden) 2009 ISBN-13 978 90 4113 115 7

MI-159 J.L.M. Gribnau, A.O. Lubbers & H. Vording (red.), *Terugkoppeling in het belastingrecht*, Amersfoort: Sdu Uitgevers 2008, ISBN 978 90 6476 326 7

MI-160 J.L.M. Gribnau, *Soevereiniteit en legitimiteit: grenzen aan (fiscale) regelgeving*, (oratie Leiden), Sdu Uitgevers 2009, ISBN 978 90 6476 325 0

MI-161 S.J. Schaafsma, *Intellectuele eigendom in het conflictenrecht. De verborgen conflictregel in het beginsel van nationale behandeling* (diss. Leiden), Deventer: Kluwer 2009, ISBN 978 90 13 06593 0

MI-162 P. van Schijndel, *Identiteitsdiefstal*, Leiden: Jongbloed 2009

MI-163 W.B. van Bockel, *The ne bis in idem principle in EU law*, (diss. Leiden), Amsterdam: Ipskamp 2009, ISBN 978 90 90 24382 5

MI-164 J. Cartwright, *The English Law of Contract: Time for Review?*, (oratie Leiden), Leiden 2009.

MI-165 W.I. Koelewijn, *Privacy en politiegegevens. Over geautomatiseerde normatieve informatie-uitwisseling*, (diss. Leiden), Leiden: Leiden University Press 2009, ISBN 9 789087 280703

MI-166 S.R.M.C. Guèvremont, *Vers un traitement équitable des étrangers extracommunautaires en séjour régulier. Examen des directives sur le regroupement familial et sur les résidents de longue durée*, (diss. Leiden), Zutphen: Wöhrmann Printing Service 2009, ISBN 978 90 8570 419 5

MI-167 A.G. Castermans, I.S.J. Houben, K.J.O. Jansen, P. Memelink & J.H. Nieuwenhuis (red.), *Het zwijgen van de Hoge Raad*, Deventer: Kluwer 2009, ISBN 978 90 13 07029 3

MI-168 P.M. Schuyt, *Verantwoorde straftoemeting*, (diss. Nijmegen), Deventer: Kluwer 2009, ISBN 978 90 1307 156 6

MI-169 P.P.J. van der Meij, *De driehoeksverhouding in het strafrechtelijk vooronderzoek*, (diss. Leiden), Deventer: Kluwer 2010, ISBN 978 90 1407 158 0

MI-170 M.V. Polak (red.), *Inbedding van Europese procesrechtelijke normen in de Nederlandse rechtsorde*, Nijmegen: Ars Aequi Libri 2010, ISBN 978 90 6916 714 5

MI-171 E. Koops, *Vormen van subsidiariteit. Een historisch-comparistische studie naar het subsidiariteitsbeginsel bij pand, hypotheek en borgtocht*, (diss. Leiden), Den Haag: Boom Juridische uitgevers 2010, ISBN 978 90 8974-259-9

MI-172 H.H. Kielman, *Politiële gegevensverwerking. Naar een effectieve waarborging*, (diss. Leiden 2010). ISBN 978 90 8570 503 1

MI-173 K. Siewicz, *Towards an Improved regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*, (diss. Leiden 2010).

MI-174 Laurens Mommers, Hans Franken, Jaap van den Herik, Franke van der Klaauw, Gerrit-Jan Zwenne (red.) *Het binnenste buiten.* (Liber amicorum prof. mr. A.H.J. Schmidt Leiden). Leiden: eLaw@leiden 2010.

Zie voor de volledige lijst van publicaties: www.law.leidenuniv.nl/onderzoek