Inf Syst Front
DOI 10.1007/s10796-009-9180-1

Managing license compliance in free and open source software development

G. R. Gangadharan · Vincenzo D'Andrea · Stefano De Paoli · Michael Weiss

Abstract License compliance in Free and Open Source Software development is a significant issue today and organizations using free and open source software are predominately focusing on this issue. The non-compliance to licenses in free and open source software development leads to the loss of reputation and the high costs of litigation for organizations. Towards an automated compliance management, we use the Open Digital Rights Language to implement the clauses of open source software licenses in a machine interpretable way and propose a novel algorithm that analyzes compatibility between free and open source software licenses. Also, we describe a framework that inductively manages compliance of license clauses in a free and open source software development. We simulate and evaluate the formalized license compliance management by analyzing a real-time open source software project GRASS.

Keywords License compliance · Free and open source software · Compliance management · Rights expression languages · Compatibility analysis

G. R. Gangadharan (✉)
Novay, Enschede, The Netherlands
e-mail: geeyaar@gmail.com, gr@novay.nl

V. D'Andrea
University of Trento, Trento, Italy

S. De Paoli
National University of Ireland Maynooth,
Maynooth Co. Kildare, Ireland

M. Weiss
Carleton University, Ottawa, Canada

1 Introduction

Compliance is a state of accordance with certain established guidelines, internal policies, or legislation. Several factors participate in increasing the focus on compliance. The first, and often foremost, is to consider the fact that, in recent years, financial scandals related to large companies prompted for new laws, focusing explicitly on compliance requests to companies and organizations. The Sarbanes-Oxley Act of 2002[1] is an example of legislation organizations in the U.S. need to comply with. The regulations set forth in the Sarbanes-Oxley Act cover the issues of auditor independence, internal control assessment, and enhanced financial disclosure. All U.S. public company boards, management, and public accounting firms are required to be compliant with the Sarbanes-Oxley Act, failure of which leads to legal punishments.

A second factor is related to the increased relevance and complexity of internal procedures and Information Technology (IT) within organizations, especially in the context of mergers and acquisitions of companies. The Control Objectives for Information and related Technology[2] (COBIT) provided by the Information Systems Audit and Control Association (ISACA), and the IT Governance Institute (ITGI) is an example for a set of guidelines for developing appropriate IT governance and control in an organization. An organization practicing these guidelines, according to the claims of COBIT, can maximize the benefits derived from the use of IT. An organization can have a set of internal policies, for

[1] United States Pub. L. No. 107-204, 116 Stat. 745.
[2] www.isaca.org/cobit.htm

Published online: 21 May 2009

instance, for supply chain management. It is expected to comply with these policies when purchasing materials.

In several cases, business compliance is audited on a per-case basis. Several researches progress in managing and measuring business compliance in organizations (Governatori et al. 2006). A static compliance checking approach is presented in Liu et al. (2007) to model processes and compliance issues. Managing and monitoring business compliance concerns are addressed by Giblin et al. (2006), Ghose and Koliadis (2007). However, the few studies that exist in the area of license compliance in information systems are still at an early stage.

A third factor to consider is the augmented relevance of Free and Open Source Software (FOSS) in Information Systems development, hence the need to carefully consider the fulfillment of licenses' clauses. Providing a framework to automatically enforce compliance to such licenses, guidelines, policies or legislation would be "ideal". However, this is highly difficult due to the "rich texture" of legal documents and guidelines.

Free and open source software is widely used by organizations and individuals and viewed as a new approach to developing software, both in the case of FOSS development and in the case of the development of standard, proprietary software which could rely on FOSS code or methodologies (Ruffin and Ebert 2004). New software can be developed by integrating FOSS components or incorporating source code fragments, thus adding value in terms of functionality and quality. The use of FOSS components in developing a new software requires developers to comply with the terms of the licenses associated with those components. The issues related to this compliance scenario are of paramount importance, because the license of a FOSS component can impact the whole Information System or computer application being developed. For instance, in the case of the GPL license,[3] a component with this license would require the whole program using the component to be released under the same terms. In addition, compliance with FOSS licenses can be complex due to the following reasons:

- Licenses vary in the privileges and restrictions imposed on a licensor regarding use, modification or redistribution of the FOSS.
- License clauses can be unacceptable to users, or can be incompatible with other licenses used in the same software.
- Choosing different licenses during different phases of the development of a software project can be quite confusing and create incompatibilities of licenses.

Managing compliance with FOSS licenses is essential to prevent the inadvertent dilution of authors' rights in an information system development. An automated approach is preferred to verifying license compliance of an FOSS being developed. The work presented in this paper is built upon our previous research on service licensing (Gangadharan et al. 2007b). We revise and extend the approach according to the specific needs of FOSS licenses. In this paper, we will argue for a machine interpretable form of FOSS licenses and develop a framework for managing license compliance in a FOSS development process. The salient contributions of our paper are as follows.

- A comprehensive framework towards managing license compliance in FOSS based on analyzing textual patterns of FOSS licenses and following the guidelines of compatibility analysis by the formalized license compliance (FLC) algorithm.
- The study of GRASS[4] development project and verification of compliance of license terms by simulation of the proposed FLC algorithm.

In Section 2, we present a real scenario of compliance issues faced in a project familiar to us. Section 3 conceptualizes various compliance issues in FOSS development. Section 4 analyzes the best practices for compliance in FOSS development today, highlighting the insufficiencies of those approaches. In Section 5, we describe an algorithm for formalized license compliance as an automated way of managing compliance. Based on the algorithm in Section 5, Section 6 illustrates a simulated evaluation of GRASS scenario where the candidate licenses are analyzed for compliance. In Section 7, we present a framework that inductively manages compliance of license clauses in a FOSS development.

2 A motivating scenario: GRASS

In order to illustrate common compliance issues in software development projects, we propose an example: that of the Geographical Information System known as Geographic Resources Analysis Support System (GRASS). GRASS started in 1982 as a small development project of the United States Army Corp of

[3] www.gnu.org/licenses/gpl.html

[4] http://grass.osgeo.org/

Engineers Research Laboratory (USA-CERL). During the years the system grew and in 1996 GRASS was already a mature project, with a source code base of approximately 300,000 lines. A further characteristic of GRASS was its open nature. Due to the provisions of the article 105 of the United States Copyright Act, the system was distributed as a Public Domain (PD) software.[5] This feature made GRASS, from its beginning a particular type of FOSS.[6]

The successful story of GRASS turned soon into a debacle. In 1996 USA-CERL decided to stop the development of the system and at this point GRASS entered in a phase of stagnation.

In 1998, however, a new development team (known as GRASS Development Team—GDT) was formed with the purpose to re-launch the development of the system (De Paoli and D'Andrea 2008). The GDT was—and it is still—composed of international voluntary researchers not related in any way with the Army.

Interestingly the passage from latest USA-CERL PD GRASS (Version 4.1) to the new GDT GRASS (Versions 4.2, 4.2.1, 5.0) marked a phase of uncertainty for the Copyright ownership of the software. The GDT had the following major problems.

1. to manage the Copyright of a US Army software
2. to protect their new additions on GRASS

In May 1999, a discussion on these problems took place in the GRASS users' mailing lists. The main problem addressed by the GDT was to decide whether it was legal to add copyright to USArmy PD software and release GRASS under a software license. Moreover, one of the concerns of the GDT was to adopt a well known FOSS license, which could have allowed an easy adoption of GRASS by the—at that time—emerging Linux community.

Interestingly, the GDT found a simply and smart solution: the new versions of GRASS were considered as derivative works of art based on the original USA-CERL GRASS. In October 1999, the new versions of the system (GRASS 5.0 onwards) were re-released under the terms of the GPL (Free Software Foundation 1991). Despite the fact that GRASS was distributed by the Army as PD software, many modules of GRASS were in fact owned by their developers. This was due to the open nature of the PD GRASS, which attracted several free-lance contributions during its life. Many of these modules were fully integrated in GRASS with their own copyright statements and licenses. The GDT faced then the need to comply with the copyright of such modules. In December 1999, the GDT started an internal review process in order to verify the existence of "contaminated" code within the newly GPL licensed GRASS. The following message, taken from the GRASS Developers Mailing List, well illustrates the problem:

> *Searching for the keyword "commercial" I turned up a couple of problematic cases.* ⋯
>
> ***Permission to use***, copy, modify, and distribute this software and its documentation ***for non-commercial purposes is hereby granted***. This software is provided "as is" without express or implied warranty.

The internal review was done searching problematic cases thorough the code. The above clause (in boldfaced italics), for example, seemed to be incompatible with the GPL—prohibiting the integration with GRASS—due to the permission to use the code only for *non-commercial purposes*. The GPL, in fact, clearly allows the commercial use of software, in particular its term 1 states that "*You may charge a fee for the physical act of transferring a copy.*" In the above example, the conflict between the GRASS code and the GPL was solved by the GDT asking the original developer for permission to re-release his commands under the GPL.

The above example illustrates one of the strategies adopted by the GDT to solve a license compliance. However not every contaminated software could be turned by the GDT into GPL software. For instance USA-CERL had a special agreement to freely use three small Numerical Recipes (NR)(Press et al. 2007) subroutines within GRASS. When GRASS was released under the GPL, the existence of NR code became a problem. The following is the Copyright clause that was attached to the NR in GRASS:

> Based on "Numerical Recipies in C: The Art of Scientific Computing" (Cambridge University Press, 1988). *Copyright (C) 1986, 1988 by Numerical Recipes Software.* **Permission is granted for unlimited use within GRASS only.**

The permission to use NR was granted only for GRASS without, for example, the possibility to transplant the code into other programs. This was considered by GRASS developers as a clear conflict with the GPL, which provides instead users with the freedom to share the software. In this case the GDT was forced to eliminate the NR derived code from GRASS code

[5]The article 105 states that works of art realized by US public institutions—such as the Army—are not eligible to copyright protection and therefore fall in the Public Domain.

[6]The public domain license is used by 0.6% of all FOSS projects listed on ohloh, a popular directory of FOSS projects.

and to provide the users with new and GPL compatible solutions.

Interestingly the license compliance problems faced by the GDT had not found a conclusion with the internal review made after GRASS release with the GPL. During the years several others conflicts emerged, making the issue of compliance one of the major concerns of the developers.

An interesting problem arose with a visualization and animation tool for GRASS data known as NVIZ. The internal review process of the GRASS development team detected the incompatibility of the NVIZ license with the GPL. What follows is a passage of the NVIZ license that renders it incompatible with the GPL:

> The visualization library and programs, both binary and source is copyrighted, but available without fee for education, research and non-commercial purposes.

Again, what we have here is the problem related to the purpose of being commercial or non-commercial. NVIZ, according to its license, was free to be used but only for non-commercial purposes. Being NVIZ a tool specifically developed for GRASS, the original authors took the decision to release NVIZ as free software under the provisions of the GNU GPL.

DWG Autocad data format is one of the many formats that modern geographical information systems need to handle. In order to provide GRASS with DWG import/export functionalities the GDT decided to use a library known as OpenDWG.[7] Suddenly, the GDT realized that the OpenDWG license was not compatible with the GPL, due to the limits to use OpenDWG for commercial purposes:

> *Linking to the libraries under GNU GPL with OpenDWG and distributing the binaries is a violation of the GNU GPL. The person doing this looses the right to use or distribute that library.*

At compilation time, OpenDWG and some portion of GRASS become in fact a derivative work, hence creating the conditions for a GPL violation. The OpenDWG library was then considered useless for the purpose of providing GRASS users with DWG functionalities.

Finally as a last example of compliance we look at Contributor Agreements. In 2006, GRASS joined the Open Source Geospatial Foundation[8] (OSGeo) as one of the founder projects. In order to become a member of the OSGeo there was the request for each developer with CVS/SVN commit access to sign the OSGeo Contributor License Agreement (CLA). The OSGeo CLA—which has not been finalized yet—had not required developers to transfer the ownership of copyright to the foundation, but required to provide the OSGeo with a license to use the code for creating collective works. It is interesting to observe the following clause from an early draft of the CLA:

> 2. ⋯ You hereby grant to the Foundation and to recipients of software distributed by the Foundation a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, sublicense, and distribute Your Contributions and such derivative works.

The previous was considered by many GRASS developers as an unacceptable license clause:

> This appears to suggest that the foundation can redistribute contributions under whichever license it chooses, [⋯] all contributions will essentially be under a BSD/MIT style license (i.e. permitting the creation of proprietary derivatives).

Since GRASS is distributed under the terms of the GPL, many developers clearly stated their intention to reject to sign the CLA due to the possibility for the foundation to modify the license (choosing a non-copylefted one), hence opening what was considered a problematic issue: the creation of proprietary derivative work from GRASS code.

3 License compliance issues in FOSS development

The exploration of the GRASS development project illustrates several kinds of compliance issues. These issues are not specific to FOSS licenses but the choice to adopt or develop specific software for an organization often requires to be compliant with licenses.

Following are the descriptions of FOSS licensing clauses that are commonly referred in this paper.

Composition: By composition, we refer to a property of a FOSS component that allows itself to be used in association with another software component. The operations of a composite software includes a value addition that is provided by the software being composed.

Attribution: A FOSS component may expect attribution (a kind of moral right) for its use by other software components. Attribution signifies a decent sign

[7] http://www.opendwg.org/

[8] http://www.osgeo.org

of respect to acknowledge the creator. Many licenses consider attribution as the most basic of requirements made by a license.

Sharealike: A FOSS software could require another component to follow its same licensing terms and conditions. The requirement of a software to follow the same licensing terms is known as Copyleft of GPL or Sharealike of Creative Commons (Lessig 2004). Sharealike imposes the requirement that any work altered, transformed, or built upon the original work having sharealike clause must be distributed on exactly the same terms. Thus, any full copyleft license automatically becomes a sharealike license. But, many sharealike licenses are only partial copyleft licenses.

Non-Commercial Use: A FOSS component can be used either for commercial purposes or for non-commercial purposes. By including the clause of non-commercial use, a FOSS component denies its use for commercial purposes. We follow the best practice guidelines proposed by Creative Commons to clarify the meaning of non-commercial use of a software.

In general, in a FOSS development, license conflicts arise in the following scenarios.

Conflicts caused by unacceptable license clauses A license may contain certain clauses that are unacceptable to a software author. The cause for unacceptance is simply individual choice.

This is a clause on distribution from the Lucent Public License Version 1.0:

> *While this license is intended to facilitate the commercial use of the Program,* ***the Distributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for Contributors****. Therefore, if a Distributor includes the Program in a commercial product offering, such* ***Distributor (*****"Commercial Distributor"****) hereby agrees to defend and indemnify*** *every Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions ... with its distribution of the Program in a commercial product offering.*

Said terms require the distributor to defend each contributor. If a distributor does not wish to follow this clause, the license becomes unacceptable for them.

Conflicts caused by incompatible license clauses Certain clauses of a license can directly prohibit integration of an open source software component distributed with certain other license.

For example, the Apache License Version 2.0 is not compatible with the GPL Version 2.0 due to the following patent clauses on the Apache license:

> *Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted.*

However, the Apache License Version 2.0 is considered as a free software license (based on the definition of the Free Software Foundation) and is compatible with the GPL Version 3 by the clause 11 of the GPL Version 3:

> *A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.*

Conflicts caused by changing licenses between releases The software organization should be careful in selecting a license for releasing a particular version of the software because the license of a particular release can have direct impact on future releases.

Consider a FOSS component S_A released under the GNU General Public License (GPL) license. At some point in the future, the licensor may decide to release a new version S_A under two different licenses say, GNU GPL[9] and Affero GPL.[10] However, the Affero GPL is incompatible with GNU GPL version 2 because of Section 2(d) that covers the distribution of application programs via web services or computer networks:

> *If the Program as you received it is intended to interact with users through a computer network and if, in the version you received, any user interacting*

[9] http://www.gnu.org/copyleft/gpl.html

[10] http://www.affero.org/oagpl.html

with the Program was given the opportunity to request transmission to that user of the Program's complete source code, you must not remove that facility from your modified version of the Program or work based on the Program, and must offer an equivalent opportunity for all users interacting with your Program through a computer network to request immediate transmission by HTTP of the complete source code of your modified version or other derivative work.

Thus, the release of S_B under Affero GPL conflicts with the license of the previous version S_A. However, the GPL version 3 and the GNU AGPL version 3[11] are compatible.

4 License compliance: An analysis of current practices

In recent years, an increasing number of law suits (BusyBox vs. Monsoon Multimedia Inc., Netfilter/iptables vs. Sitecom, The SCO Group vs. Linux, and similar other trials (Hassin 2007)) have been filed involving several issues of compliances in FOSS. Analyzing these cases reveal the need for:

- a better interpretation and enforceability of FOSS licenses that highlight the significance of compliance.
- sanctions in case of failure to comply.

These cases also pointed to the high costs of litigation for non-compliance. Many organizations are, therefore, trying to establish policies on the inclusion and verification of the presence of use and that allow them to verify the presence of third-party components in a proprietary code base.

Organizations have since developed metrics for managing compliance in FOSS development. Below we discuss some of the best practices for compliance to licenses currently in use (Fan et al. 2004).

Contributors Agreement A FOSS project may require a way of confirmation from its contributors through agreements that the author of the source code ensures the cleanliness of the code. With this effort, the project can be expected to produce a codebase that has clear IP provenance and protects the IP rights of others. The contributors of Apache Harmony Project (supported by the Apache Software Foundation) are required to sign a contribution checklist[12] not only to ensure the cleanliness of the code but also to encourage the contributors to carefully examine their contributions before bringing to the project. At the Eclipse Foundation,[13] two levels of legal documentation are currently in use to cover all contributions of source code made by developers (Campbell 2007). The Eclipse Foundation requires that all contributions are made by the rightful copyright holder and under the Eclipse Public License[14] (EPL). A committer agreement[15] is signed by each committers to stipulate their contributions as their original work. If a committer is sponsored to work on an Eclipse project by a Member organization, then that organization is asked to sign a Member Committer Agreement[16] to ensure the intellectual property rights of the organization are contributed under the EPL. Furthermore, the Eclipse Foundation ensures that submissions through Eclipse web page are licensed to others under the terms of the Eclipse Foundation.

Internal Review At every release and build of a FOSS project, organizations should verify whether any contaminated code is used in the software. A set of team members can verify the cleanliness of the source code in a project. The team can also verify that no unapproved modifications were made to external software components.

Compliance Tools Companies such as BlackDuck[17] and Palamida[18] offer products for ensuring IP compliance. These products compare the inputted source code against a knowledge base built from an assortment of FOSS projects and report matches between the inputted code and code in the knowledge base. However, we cannot evaluate these products as ideal solutions because these solutions fail to address formalization of licenses and license conflicts. Furthermore, it is highly difficult to verify the other kinds of IP infringements (such as patent and trademarks violations) made by the code.

The Fossology project[19] (an internal development effort by Hewlett Packard Company) proposes to analyze all of the source code for a given project and intelligently report all of the licenses being used, based

[11] http://www.fsf.org/licensing/licenses/agpl-3.0.html

[12] http://harmony.apache.org/bulk_contribution_checklist.txt

[13] http://www.eclipse.org/org/

[14] http://www.eclipse.org/legal/epl-v10.html

[15] http://www.eclipse.org/legal/committer_process/EclipseIndividualCommitterAgreementFinal.pdf

[16] http://www.eclipse.org/legal/EclipseMemberCommitterAgreementFinal.pdf

[17] http://www.blackducksoftware.com

[18] http://www.palamida.com

[19] http://fossology.org/

on the license declarations and tell-tale phrases that identify software licensing (Gobeille 2008).

One of the research and development domains of the Quality Platform for Open Source Software (Qualipso) project[20] proposes to provide guidelines and tools to facilitate an intellectual property tracking process with open source, and defining a coherent family of FOSS licenses, compliant with national laws and European regulations.

In academia, to the best of our knowledge, there is an obvious paucity of research on the topic of IP compliance associated with FOSS. A compliance tool described in Nordquist et al. (2003) gives an automated way to help software developers in detecting license conflicts. However, the scope of this tool is very limited and immature. Some informal and unstructured discussions about the concerns of IP and FOSS are explicated in the forum of Open Business Readiness Rating.[21]

As there are no existing standards for the verification of compliance, the application of present best practices is subject to individual organizations.

5 Formalized license compliance in FOSS

Determining the compatibility of FOSS licenses in a FOSS project is one of the most challenging and a mandatory step to ensure compliance with license terms. Compatibility analysis is a process of matchmaking of candidate open source component licenses (at license clause level) in developing a new software. The matchmaking algorithm outlined below performs a compatibility analysis between any two given licenses and decides whether the licenses are compatible. A license is compatible with another license if all license clauses are compatible. The given candidate components can be combined, if their license are found compatible by the algorithm.

Open Digital Rights Language (*ODRL*) (Iannella 2002) is an open standard language for the expressions of terms and conditions over assets, in open and trusted environments. Basically, the ODRL specification is presented in two sections as follows.

- ODRL Expression Language: Defines basic terms of rights expressions and the organization of these terms, often using abstract concepts.
- ODRL Data Dictionary: Defines the semantics of the concrete terms which are used to express an instance of a rights specification.

[20] http://www.qualipso.org

[21] http://www.openbrr.org/forums/viewtopic.php?t=104

ODRL is based upon an extensible model for rights expression which comprises following core entities and their relationships.

- *Assets:* Resource being licensed (identified uniquely).
- *Rights:* Rules concerning permissions (the actual usages or activities allowed over the assets), constraints (limits to these permissions), requirements (the obligations needed to exercise the permission), and conditions (the specifications of exceptions that, if become true, expire the permissions and renegotiation may be required).
- *Parties:* Information regarding the service provider, consumer, broker etc.,

With these three entities, ODRL expresses offers (proposals from rights holders for specific rights over their assets) and agreements (contracts or deals between the parties, with specific offers). Together these core entities, ODRL allows for a wide and flexible range of expressions to be declared.

Although ODRL is a right expression language for specifying rights over digital assets, we can use it for expressing a software license in machine interpretable way. A machine interpretable representation of an open source software license presented in this paper is not a substitute for a legal license and is an endeavor for machine interpretable expression of open source licenses. The representation of FOSS licenses and the matchmaking algorithm are based on well-established results presented in our earlier work in the field of service oriented computing (Gangadharan et al. 2007b). The illustrated FLC algorithm in this paper explicitly determines the compatibility between some of the prominent clauses of FOSS licenses.

A license L_S in ODRL for a software S consists of a finite set of models (generally referred as license clauses), each of which further consists of a set of elements. Elements can be specified with value or without value (empty element having the element type only). Elements can contain other elements that can give rise to an arbitrarily deep hierarchy of elements within elements.

Assuming that semantics inside a license are agreed by software developers and consumers, a simple formalized license compliance (FLC) algorithm for matching a license L_α (for a software α) with another license L_β (for a software β) is given as follows. We use the symbol $\rightleftharpoons$ to denote compatibility. We denote a model in L_α as m_α. We refer to the elements of a model in a software α as $Elements(m_\alpha)$ and the elements nested inside an element as $Elements(e_\alpha)$. Our algorithm analyzes not only the compatibility between a model or

Table 1 Rules for determining compatibility with unspecified licensing elements

Specified clause	Compatibility	Rationale
Composition	*Incompatible*	A license denying composition cannot be compatible with a license allowing composition.
Attribution	*Compatible*	The requirement to specify attribution will not affect the compatibility when unspecified.
Sharealike	*Compatible*	The composite license must be similar to the license with the Sharealike clause.
Noncommercialuse	*Incompatible*	Commercial use is denied by non-commercial use.

an element with the corresponding model or element in licenses but also the case of unspecification of a model or an element (referred as $unspecified(m_\alpha)$ or $unspecified(e_\alpha)$ for a software α).

In certain cases, the absence of one or several of these clauses will not cause non-compliance issues. Table 1 lists rules to determine the compatibility of license elements with unspecified ("don't care") license elements (based on Open Source Initiative 2006 and Free Software Foundation 2009).

Two licenses are compatible, if all the respective models in both the licenses are compatible.

$$
\begin{aligned}
(L_\alpha \rightleftharpoons L_\beta) \quad &\Longleftrightarrow \\
(\forall m_\alpha : m_\alpha \epsilon L_\alpha \quad (\exists m_\beta : m_\beta \epsilon L_\beta \quad &\Rightarrow \quad (m_\alpha \rightleftharpoons m_\beta)) \\
&\vee \quad unspecified(m_\alpha)) \\
\wedge \quad (\forall m_\beta : m_\beta \epsilon L_\beta \quad (\exists m_\alpha : m_\alpha \epsilon L_\alpha \quad &\Rightarrow \quad (m_\alpha \rightleftharpoons m_\beta)) \\
&\vee \quad unspecified(m_\beta))
\end{aligned}
$$

A model m_α is compatible with another model m_β, if the model types are same (represented by $\equiv$) and their elements are compatible.

$$
\begin{aligned}
(m_\alpha \rightleftharpoons m_\beta) \quad &\Longleftrightarrow \\
&(m_\alpha \equiv m_\beta) \\
\wedge \ (\forall e_\alpha : e_\alpha \epsilon Elements(m_\alpha) \quad (\exists e_\beta : e_\beta \epsilon Elements(m_\beta) &\Rightarrow \\
&(e_\alpha \rightleftharpoons e_\beta)) \quad \vee \quad unspecified(e_\alpha)) \\
\wedge \ (\forall e_\beta : e_\beta \epsilon Elements(m_\beta) \quad (\exists e_\alpha : e_\alpha \epsilon Elements(m_\alpha) &\Rightarrow \\
&(e_\alpha \rightleftharpoons e_\beta)) \quad \vee \quad unspecified(e_\beta))
\end{aligned}
$$

Now, an element e_α is compatible with another element e_β, if e_α and e_β have same type (represented by $\equiv$) or e_α and e_β have equal value. Furthermore, for all nested elements, corresponding elements are compatible. The algorithm refers to the Table 1 to decide compatibility in case an element in a given license is unspecified.

$$
\begin{aligned}
(e_\alpha \rightleftharpoons e_\beta) \quad &\Longleftrightarrow \\
&(e_\alpha \equiv e_\beta) \\
\wedge \quad &(Value(e_\alpha) = Value(e_\beta)) \\
\wedge \ (\forall e_\alpha : e_\alpha \epsilon Elements(e_\alpha) \quad (\exists e_\beta : e_\beta \epsilon Elements(e_\beta) &\Rightarrow \\
&(e_\alpha \rightleftharpoons e_\beta)) \quad \vee \quad unspecified(e_\alpha)) \\
\wedge \ (\forall e_\beta : e_\beta \epsilon Elements(e_\beta) \quad (\exists e_\alpha : e_\alpha \epsilon Elements(e_\alpha) &\Rightarrow \\
&(e_\alpha \rightleftharpoons e_\beta)) \quad \vee \quad unspecified(e_\beta))
\end{aligned}
$$

6 Simulation of license compliance in the GRASS scenario

In this section, we illustrate and evaluate the FLC algorithm by applying it to the GRASS scenario. We have represented some of the licenses of softwares in ODRL that are involved in the GRASS and simulated the FLC algorithm over the GRASS development scenario as described in Section 2.

As GRASS is licensed under GPL, we describe a partial representation of a GPL license in ODRL as follows. We represent the Copyleft clause of GPL as *sharealike* of ODRL Creative Commons Profile (Iannella 2005) and the *indemnity* clauses by ODRL Service Licensing Profile (Gangadharan et al. 2007a).

```
  <!-- Namespace declarations go here-->
1   <o-ex:offer>
2    <o-ex:requirement>
3       <o-cc:sharealike/>
4    </o-ex:requirement>
5    <sl:indemnity>
6       <sl:thirdpartyinfringementsclaims/>
7    </sl:indemnity>
8  </o-ex:offer>
```

Following is a set of license clauses of NVIZ visualization library and programs used in GRASS.

> The visualization library and programs, both binary and source is copyrighted, but available without fee for education, research and non-

> commercial purposes. Users may distribute the binary and source code to third parties provided that the copyright notice and this statement appears on all copies and that no charge is made for such copies. Any entity wishing to integrate all or part of the source code into a product for commercial use or resale, should contact the authors of the software.

The equivalent ODRL representation of this fragment is as follows.

```
1   <o-ex:offer>
2    <o-ex:requirement>
3       <o-cc:attribution/>
4    </o-ex:requirement>
5    <o-ex:constraint>
6      <cc:NonCommercialUse/>
7    </o-ex:constraint>
8  </o-ex:offer>
```

The FLC algorithm compares the license clauses of NVIZ visualization library and program with the license clauses of GRASS. As the models given in both the licenses (line 2) are the same (`<o-ex:requirement>`), the algorithm verifies the elements. Though the `<o-cc:sharealike/>` clause in the license of GRASS is unspecified in the license of NVIZ, these licenses are compatible. The rationale for this compatibility is that `sharealike` affects the composite license requiring that the composite license should be similar to the license having `sharealike` element. Similarly the unspecification of infringement and attribution clauses in any of the licenses does not have direct impact on compliance of licenses (see Section 7). The `<o-ex:constraint>` model of NVIZ license (in lines 5,6, and 7) specifies the element `<o-cc:noncommercialuse>`. When the FLC algorithm looks for the element `<o-cc:noncommercialuse>` in the GRASS license, the algorithm is unable to find as the element is unspecified. This indicates that the GRASS can be used for commercial purposes. From Table 1, the algorithm finds that these clauses are incompatible, and thus the licenses become incompatible.

The Clause 2 of the Numerical Recipes (NR) Institutional Subscriber license states as follows.

> ... you may, under this license, transfer precompiled, executable applications incorporating the code to other, unlicensed, machines or persons, providing that
>
> (i) the application is noncommercial (i.e., does not involve the selling or licensing of the application for a fee), and
>
> (ii) the application was first developed, compiled, and successfully run by you, and
>
> (iii) the code is bound into the application in such a manner that it cannot be accessed as individual routines and cannot practicably be unbound and used in other programs. That is, for the terms of this paragraph to apply, your application user must not be able to use Numerical Recipes code as part of a program library or "mix and match" workbench.

Suppose you create a new application, say a spreadsheet, based (also) on a function F in NR where F is covered by a license L. Besides being used as a "normal" standalone program, your application could allow a programmer to invoke some of your spreadsheet operations as if they were elements in a software library. This could be done, for instance, as objects methods invokable by external programs or by services made available over the web. In this case, a license denying composition would not allow you to include the software covered by that license in the parts of your application that you make available for other programs use. In this case, since L denies composition it does not allow you to include F in the parts of your application that you make available for other programs use.

The Numerical Recipes license clause is represented in ODRL as follows.

```
1   <o-ex:offer>
2    <o-ex:constraint>
3      <cc:NonCommercialUse/>
4    </o-ex:constraint>
5  </o-ex:offer>
```

Applying the FLC algorithm between the licenses of GRASS and Numerical Recipes (NR), we see that the NR license does not have an `<sl:composition>` clause (the license clause that allows the composition of components). Hence, the FLC algorithm does not permit the NR code to be mixed with GRASS. As we have seen in Section 2, the permission to use NR was granted only for GRASS without the possibility to transplant the code into other programs. As GRASS is licensed under GPL, keeping the NR code in GRASS causes non-compliance.

The causes of non-compliance of certain licenses in the GRASS scenario can be of two kinds:

1. As GRASS is licensed under GPL, GRASS does not deny the commercial use. However, licenses denying the commercial use cannot be compliant with GRASS.

2. If a software license does not allow itself to be combined with any other software, by default, GRASS becomes non-compliant to this software.

We found that by following the FLC algorithm, we are able to discover the same non-compliance to licenses in GRASS which the GDT had found during the development of GRASS (detailed in Section 2). Thus, we can evaluate the validity of the FLC algorithm by simulation of licenses in the GRASS scenario.

7 Formalized license compliance management framework

In this section, we present the foundations of a framework that analyzes and implements FLC algorithm. Our framework is not upfront; but it is an inductive way to address the issues of compliance analysis in FOSS development.

7.1 Transforming FOSS licenses

As some legal doctrines are inherently flexible and vague, the translation of legal concepts into a machine interpretable language is highly complex. Classically, rights over digital assets are expressed using right expression languages. However, to represent license terms of a free/open source software or component in a machine interpretable way (in a best possible manner with unambiguity), we follow a novel approach based on analyzing textual patterns of FOSS licenses (Kaminski and Perry 2007; Gangadharan et al. 2008). Understanding different contexts, forces, and solutions help us to identify the license clauses that can be expressable in ODRL.

7.1.1 Restrictive pattern

Context The software should allow free use of source code adaptable by anyone and redistributable on the same terms by anyone. Modified code of that software requires the same licensing for distribution.

Forces Users should be able to modify a software, or derive new software from the software. However, in order to avoid license forking, we would like to prevent that the new software is licensed differently from the parent software. In this way, the value-added by the changes can benefit the whole community created around the software. This benefit needs to be balanced against the need of software developers to generate profit from their software offerings.

Solution A software license with a copyleft clause (of GPL) requires value-added software to be licensed under the same terms and conditions. These clauses prevent others from turning value-added software into closed software, if the parent software is open.

7.1.2 Permissive pattern

Context A software license should allow the usage of services for any purposes, without placing any constraints as requirements.

Forces Users should be able to modify a software, or derive new software from the software. However, the source code that creates the value addition can be totally kept private.

Solution A software license (similar to BSD or MIT licenses) will not place any restrictions over execution including non-commercial uses.

7.1.3 Flexible pattern

Context The software should allow free use of source code on the same terms by anyone. Furthermore, the software could allow integration with proprietary software.

Forces A developer can want users to incorporate the developed FOSS component into a proprietary program. However, the developer requires to preserve the openness and freedom of the developed component.

Solution Having GNU Lesser General Public License styled terms allow the components to be incorporated with proprietary software.

7.2 Compatibility analysis of FOSS license clauses

The issue of compatibility arises when a new software is being developed by combining source codes having different licenses. When a FOSS component/software is combined with another FOSS component/software, their licenses are also composed. If results in incompatibilities, then the corresponding components cannot be combined. The composite license can contain licensing clauses, which need not be present in the licenses of the software that are combined.

Assume that a new FOSS component R is developed by combining FOSS components P and Q. $L(P)$, $L(Q)$, and $L(R)$ are the licenses of candidate components and $LC(P)$, $LC(Q)$, and $LC(R)$ are the set of license clauses of these candidate components. It is expected that $L(P)$ and $L(Q)$ are compatible, which, in turn, requires their license clauses to be compatible.

The FLC algorithm analyzes the compatibility of licenses at the element level. Any two software components can be combined, if the licenses of these components are found compatible by the FLC algorithm. Following are the first set of guidelines in compatibility analysis of FOSS license clauses. We do not claim that these guidelines are exhaustive.

1. A FOSS component license requiring `Attribution` (copyright notice) can be compatible with another FOSS component license that may not require `Attribution`. The requirement for specification of attribution will not affect the compatibility when it is unspecified.
2. A GPL based copyleft clause (Sharealike) affects the composite license requiring that the composite license should be similar to the license having Sharealike element. A component with GPL based clause can be combined with a component not having copyleft clause. However, the resulting component, by default, will be copylefted.
3. A FOSS license can exclusively deny to be integrated with other components. A component denying composition cannot be combined with any FOSS component or can be used (compilation or linking) by another software.

The composition of FOSS components can be represented by,

$$LC(R) = Reduce((LC_{NEW}) \cup Reduce(LC(P) \cup LC(Q)))$$

where LC_{NEW} is a set of licensing clauses exclusive to the new FOSS software. *Reduce* is an operation that eliminates duplicate clauses from the set of union of candidate license clauses.

To decide whether to include a textual (natural language) clause of a FOSS license in a machine interpretable (ODRL) representation, users can follow the following guidelines based on Bezroukov (1998):

1. An absolute requirement of a license clause is specified by the word 'MUST'.
2. A recommended requirement of a license clause is specified by the word 'SHOULD'.
3. An optional requirement of a license clause is specified by the word 'MAY'.

8 Concluding remarks

Managing license compliance in FOSS development is a significant issue today and many organizations using FOSS are focusing on this issue. In this paper, we have analyzed the causes of compliance issues through a detailed real FOSS development project. We have reviewed a set of organizational best practices currently in practice and their limitations. Towards an automated compliance management, we have proposed a novel algorithm that analyzes compatibility between FOSS licenses expressed in ODRL. We have applied and simulated our license compliance algorithm to GRASS development and evaluated the correctness of our algorithm. By inductive reasoning, a license compliance management framework is depicted as a way for users to understand and manage license compliance in developing FOSS projects.

We do not claim that our approach is generic enough to include all major FOSS licenses. However, to the best of our knowledge, our approach is one of the first attempts to manage compliance in FOSS development, in a formalized way. The main limitation of the illustrated FLC approach is related with the expression of 'obnoxious BSD advertising clause'.[22] We have defined the rules for compatibility in such a way that attribution does not affect compatibility in case of unspecification. Even we revise this, the syntax of ODRL cannot be expressed to specify the attribution levels as required by the original BSD license.

We are currently working on an approach that allows developers to specify licensing terms and select FOSS components. In this way, the conflicts caused by unacceptable license clauses can be partially automated and resolved.

References

Bezroukov, N. (1998). The idea of dynamic licensing. http://www.softpanorama.org/Copyright/License_classification/dynamic_licensing.shtml.

Campbell, J. (2007). Open source software—clarifying the IP trail. http://www.talentfirstnetwork.com.

De Paoli, S., & D'Andrea, V. (2008). How artefacts rule web-based communities: Practices of free software development. *International Journal of Web Based Communities, 4*(2), 199–219.

Fan, B., Aitken, A., & Koenig, J. (2004). Open source intellectual property and licensing compliance: A survey and analysis of industry best practices. http://olliancegroup.com/opensource/compliance_best_practices.php.

Free Software Foundation (1991). GNU general public license. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

Free Software Foundation (2009). Various licenses and comments about them. http://www.gnu.org/philosophy/license-list.html.

Gangadharan, G. R., D'Andrea, V., Iannella, R., & Weiss, M. (2007a). ODRL service licensing profile (ODRL-S). In *Proceedings of the 5th international workshop for technical, economic, and legal aspects of business models for virtual goods.*

[22] http://www.gnu.org/philosophy/bsd.html

Gangadharan, G. R., Weiss, M., D'Andrea, V., & Iannella, R. (2007b). Service license composition and compatibility analysis. In *Proceedings of the international conference on service oriented computing (ICSOC'07)*, Vienna.

Gangadharan, G. R., Weiss, M., & D'Andrea, V. (2008). Patterns for licensing web services. In *Proceedings of the European conference on pattern languages of programs (EuroPLOP)*.

Ghose, A., & Koliadis, G. (2007). Auditing business process compliance. In *Proceedings of the international conference on service oriented computing (ICSOC)*.

Giblin, C., Muller, S., & Pfitzmann, B. (2006). *From regulatory policies to event monitoring rules: Towards model driven compliance automation.* Technical report RZ-3662, IBM Research Laboratories.

Gobeille, R. (2008). The FOSSology project. In *Proceedings of the (MSR)*.

Governatori, G., Milosevic, Z., & Sadiq, S. (2006). Compliance checking between business processes and business Contracts. In *Proceedings of the 10th IEEE international enterprise distributed object computing conference (EDOC)*.

Hassin, K. (2007). Open source on trial. *Open Source Business Resource*. http://www.osbr.ca/ojs/index.php/osbr/article/view/391/352.

Iannella, R. (Ed.) (2002). Open digital rights language (ODRL) version 1.1. http://odrl.net/1.1/ODRL-11.pdf.

Iannella, R. (Ed.) (2005). ODRL creative commons profile. http://odrl.net/Profiles/CC/SPEC.html.

Kaminski, H., & Perry, M. (2007). Open source software licensing patterns. In *Proceedings of the sixth latin american conference on pattern languages of programming (SugarLoafPLoP)*.

Lessig, L. (2004). The creative commons. *Montana Law Review, 65*, 1–13.

Liu, Y., Muller, S., & Xu, K. (2007). A static compliance checking framework for business process models. *IBM Systems Journal, 46*, 335–361.

Nordquist, P., Petersen, A., & Todorova, A. (2003). License tracing in free open and proprietary software. In *Proceedings of the northwestern conference by the consortium for computing sciences in colleges*.

Open Source Initiative (2006). Open source licenses. http://www.opensource.org/licenses/alphabetical.

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (2007). *Numerical recipes. The art of scientific computing*. Cambridge: Cambridge University Press.

Ruffin, M., & Ebert, C. (2004). Using open source software in product development: A primer. *IEEE Software, 21*, 82–86.

G. R. Gangadharan is a research scientist at the Novay (formerly known as Telematica Institute), Enschede, The Netherlands. His research interests are mainly located on the interface between the technological perspective and the business perspective. His research interests include Service Oriented Computing, Internet Software Engineering, Intellectual Property Rights, Free and Open Source Systems, and Business Models for Software and Services. He can be reached at gr@novay.nl.

Vincenzo D'Andrea is an associate professor at the University of Trento, where he teaches Information Systems. His research interests include service-oriented computing, free and open source licensing, and socio-technical systems. He received his PhD in information technology from the University of Parma. He is a member of the IEEE Computer Society and the ACM. Contact him at the Department of Information Engineering and Computer Science: http://disi.unitn.it/users/vincenzo.dandrea.

Stefano De Paoli is post-doctoral researcher at the National University of Ireland Maynooth, Ireland. His research focuses on the interlinks between social sciences and computer sciences with attention to computer security, online games, free software development and online research methods. More at Stefano institutional web page: http://www.nuim.ie/nirsa/people/postdocs/stefano_de_paoli.shtml.

Michael Weiss holds a faculty appointment in the Department of Systems and Computer Engineering at Carleton University, Canada. His research interests include open source ecosystems, service-oriented architectures, mashups/Web 2.0, business process modeling, product architecture and design, and pattern languages. He can be reached at weiss@sce.carleton.ca.