QualiPSo

Quality Platform for Open Source Software

IST- FP6-IP-034763

Deliverable A1.D1.1.3

Report on Problem Scope and Definition about OSS License Compatibility

Thomas F. Gordon
Fraunhofer FOKUS, Berlin

Due date of deliverable: 31/10/2009

Actual submission date: dd/mm/yyyy

This work is partially funded by EU under the grant of IST-FP6-034763.

Change History

Version	Date	Status	Author (Partner)	Description
01.01.10	15.06.10		Tom Gordon	Minor modifications to the executive summary, abstract and introduction

Executive Summary

This Qualipso report surveys some of the legal issues which can arise when multiple software components, licensed with different Open Source licenses, are combined into collective or derivative works. A concrete scenario is used to illustrate legal issues which need to be considered by the developers of Open Source software. The basic concepts of copyright law are explained, insofar as they are relevant for license compatibility issues. The kinds of legal sources are surveyed which need to be taken into consideration and interpreted when analysing license compatibility issues, including legal principles, constitutional law, statutory copyright and contract law, case law, and various international treaties. Finally, a brief overview of legal reasoning and argumentation is provided, showing how the resolution of Open Source license compatibility issues, like all legal issues, is a creative, theory-construction process which cannot be fully well-defined and thus cannot be fully automated. Our next task will be to investigate whether methods from the field of Artificial Intelligence and Law can be applied to build tools which help developers to construct, explore and compare legal theories when analysing Open Source licensing issues.

Document Information

IST Project Number	FP6 – 034763	**Acronym**	QualiPSo
Full title	Quality Platform for Open Source Software		
Project URL	http://www.qualipso.org		
Document URL			
EU Project officer	Charles MacMillan		

Deliverable	**Number**	A1.D1.1.3	**Title**	Report on Problem Scope and Definition about OSS License Compatibility
Work package	**Number**	1,3	**Title**	Compatibility
Activity	**Number**	A1	**Title**	Legal Issues

Date of delivery	**Contractual**	31/10/2009	**Actual**	/10/2009
Status	Version 1.0, dated 9/10/2009		final ≺	
Nature	Report ☑ Demonstrator ☐ Other ☐			
Dissemination Level	Public ☐ Consortium ☐			
Abstract (for dissemination)	This Qualipso report surveys some of the legal issues which can arise when multiple software components, licensed with different Open Source licenses, are combined into collective or derivative works. A concrete scenario is used to illustrate legal issues which need to be considered by the developers of Open Source software. The basic concepts of copyright law are explained, insofar as they are relevant for license compatibility issues. The kinds of legal sources are surveyed which need to be taken into consideration and interpreted when analysing license compatibility issues, including legal principles, constitutional law, statutory copyright and contract law, case law, and various international treaties. Finally, a brief overview of legal reasoning and argumentation is provided, showing how the resolution of Open Source license compatibility issues, like all legal issues, is a creative, theory-construction process which cannot be fully well-defined and thus cannot be fully automated. Our next task will be to investigate whether methods from the field of Artificial Intelligence and Law can be applied to build tools which help developers to construct, explore and compare legal theories when analysing Open Source licensing issues.			
Keywords				

Authors (Partner)	Thomas F. Gordon (Fraunhofer FOKUS)			
Responsible Author	Thomas F. Gordon		**Email**	Thomas.gordon@fokus.fraunhofer.de
	Partner	Fraunhofer FOKUS	**Phone**	

Table of Contents

1 Introduction

In this Qualipso report we examine some legal issues which can arise when multiple software components, licensed with different Open Source licenses, are combined into collective or derivative works. Some reciprocal licenses, such as the GPL, require derivative works to be licensed under the *same* license. Clearly if multiple components are subject to different reciprocal licenses of this kind, it may be legally impossible to create a derivative work from all these components, since the license of each such component requires the same license, and no other, to be used for the entire derivative work.

Such restrictive reciprocal Open Source licenses, interpreted literally, run the risk of fragmenting Open Source software into separate islands, one for each such reciprocal license, making it much more difficult to share software and requiring a wasteful duplication of effort to develop comparable components for each license.

One way to overcome this problem is to interpret the reciprocity conditions of licenses in such a way as to allow derivative works to be licensed using any license which is *compatible* with the terms and conditions of the original license of each component, rather than just the same, identical license. This approach would raise at least two legal issues: 1) May the original license be substituted by a compatible one? And 2) Are the two different licenses in fact sufficiently compatible?

Another way to overcome this problem may be to use some works subject to reciprocal licenses in ways which do not legally create derivative works. The issue then becomes: What is the precise legal meaning of “derivative work”? What kinds of uses count as creating derivate works and which ones do not? A well-known but unresolved example of this legal issue concerns the question whether linking to a library of code creates a derivative work or only a collective work. And would it make a difference whether the library is dynamically linked, rather than statically linked?

Legal issues, not only these legal issues, cannot be answered definitely outside the context of specific legal cases in specific jurisdictions. Open Source software is copied, modified and distributed throughout the world, in particular via the Internet, but there is no single, globally applicable copyright or contract law. There are a growing number of Open Source licenses, each with their own terms and conditions. Existing Open Source licenses, such as the GPL, change over time, with new versions being issued over time. The terms and conditions of licenses need to be interpreted against the background of the relevant governing law, such as the copyright law of a particular country. There are significant differences between Common Law jurisdictions, such as the United Kingdom and the United States, and Civil Law jurisdictions such as France or Germany. Which law is governing can depend on the citizenship or residence of the licensor or licensee, the location of the claimed infringement of the terms of the license, or the terms and conditions of the license itself. The relevant laws of these jurisdictions also change over time. If the law of several jurisdictions is applicable, some method of resolving conflicts among these laws is required. To the extent that courts have interpreted relevant laws in previous cases, these court decisions will need to be taken in to account. The decisions of courts in different jurisdictions may diverge

on some legal issues, and typically are not binding on courts in other jurisdictions. There may be international treaties to consider, such as the 1886 Berne Convention for the Protection of Literary and Artistic Works, the 1952 Universal Copyright Convention, or the 1994 Agreement on Trade Related Aspects of Intellectual Property Rights (TRIPS) administered by the World Trade Organization (WTO). Some licenses may also be contracts, requiring the application of contract law in addition to copyright law. The law of contracts also varies from jurisdiction to jurisdiction. And there may be other international agreements to consider such as the 1980 United Nations Convention on Contracts for the Sale of Goods (CISG).

A further source of uncertainty is caused by the lack of a common language or terminology. There are 23 official languages in the European Union alone. Each country has its own culture, language and history, against which courts will interpret the text of statutes, licenses and contracts, even when these are written in English. There is no global *ontology* of legal concepts.

Due to this complexity, it cannot be our goal to try to answer any of these legal issues in the abstract here. Our goal is more limited, to try to more clearly define the problem and scope of Open Source license compatibility issues. What kinds of legal issues can arise in the context of the uses cases of interest to developers of Open Source software when combining software components subject to different licenses? What legal sources need to be taken into account when trying to analyse these legal issues? What kinds of legal reasoning, argumentation and problem-solving methods are relevant when making use of these legal sources to analyse these issues? In a later report we will then use this information to survey the state-of-the-art of computational models of legal reasoning and argumentation, from the field Artificial Intelligence and Law, to try to assess whether methods are available which can be applied in software tools for helping developers to analyse and understand license compatibility issues of their own projects.

The remainder of this report is organized as follows. Section 2 presents a more concrete scenario, to help make the legal issues clearer and to serve as a background for identifying use cases for software tools for assisting developers with Open Source license compatibility issues. Section 3 introduces the legal concepts and issues of interest in greater detail, by explaining the concept of a chain of title, noting the difference between a bare license and a contract, discussing sublicensing issues, contrasting academic and reciprocal Open Source licenses, discussing the distinction between collective and derivative works and explaining the relevance of these concepts and issues for analysing license compatibility issues. Section 4 surveys some of the legal sources which need to be taken into consideration and interpreted when analysing license compatibility issues, including legal principles, constitutional law, statutory copyright and contract law, case law, and various international treaties. Section 5 provides a brief overview of legal reasoning and argumentation, showing how legal problem solving is a creative, theory-construction process which cannot be fully well-defined as is thus cannot be fully automated, not even when using heuristic-search methods from Artificial Intelligence. In the law, there is never a uniquely *right* answer to some legal issue. Good arguments can always be made on both sides of any issue. Deciding legal issues requires good judgement, not just good

logic. This nature of legal reasoning leads to some necessary uncertainty and risk which cannot be entirely eliminated. This is as true for Open Source software development as for any other activity regulated by law. Section 6 concludes with a summary of the main results of this report. A bibliography of references is included in the appendix.

2 SCENARIO

To help make the Open Source license compatibility issues more concrete and comprehensible, this section presents an overview of the main components of a real Open Source project, currently being negotiated, which will develop an "argumentation toolbox" for facilitating public debates on policy issues at a European scale. Figure 1 shows the planned components of the system, along with the Open Source components upon which they depend. The licenses of the components used by the toolbox are shown in parentheses. When multiple licenses are listed for a component, the licensee is free to choose among these licenses.

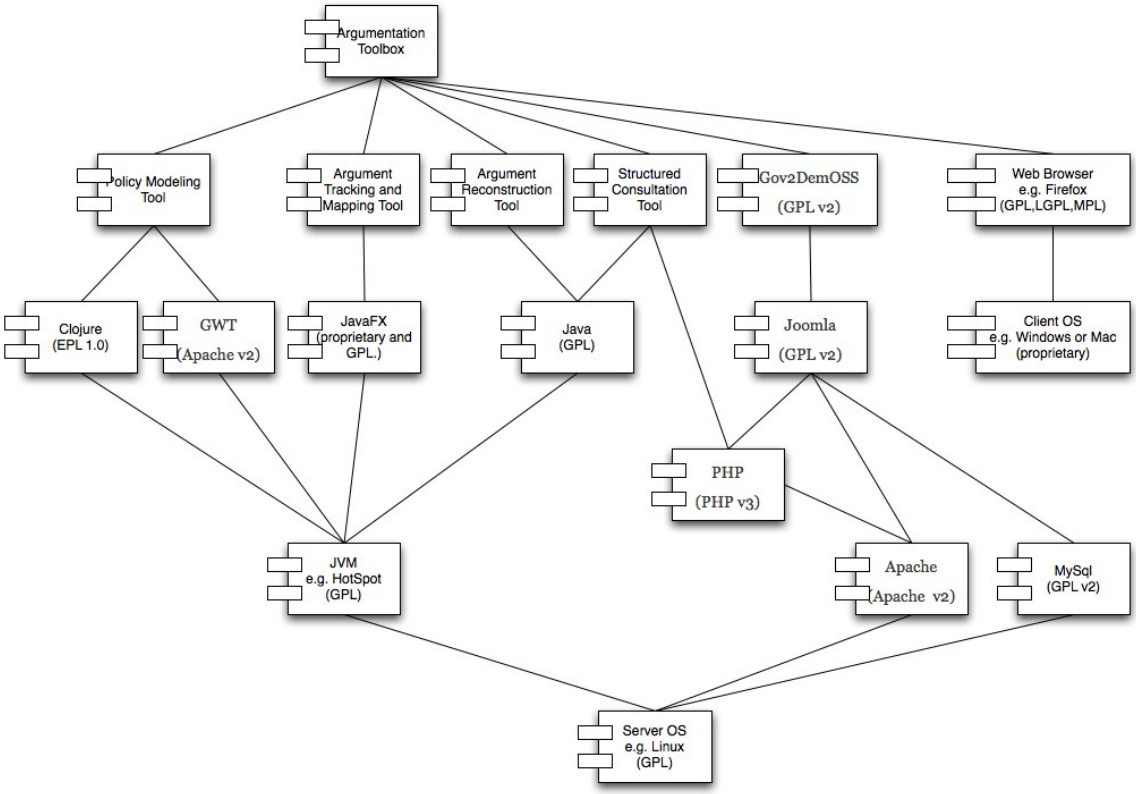

Figure 1: Argumentation Toolbox Component Dependencies

The details of the planned argumentation toolbox are not important for us here, expect to note that the system is representative of many 3-tiered, client-server applications for the World-Wide-Web. It consists of three layers: a presentation layer, application logic layer and a database layer. The presentation layer will be implemented in PHP, Java, and JavaFX. The Java code for the presentation layer uses the Google Web Toolkit (GWT), which generates JavaScript and HTML code, which are interpreted by a JavaScript engine, such as TraceMonkey and a HTML renderer, respectively. All the presentation code will run in a Web browser, such as Firefox, which provides a HTML renderer and a JavaScript runtime environment. The application logic layer uses an eParticipation platform, called Gov2DemOSS, which in turn is based on the Joomla content management

system. Joomla is implemented in PHP and depends on the Apache web server. The application logic of the various argumentation tools are implemented in Java and Clojure. Both of these languages are compiled into bytecode which runs on a Java Virtual Machine (JVM), such as HotSpot. The database layer uses a MySql relational database. The JVM, Apache web server and MySql database run a Linux server. The web browser runs on a client platform, which these days could be some mobile device, not only a conventional desktop computer. The client platform is not shown in the dependency diagram.

Several of the components are based on standards with multiple implementations, including HTML, JavaScript (actually ECMAScript), and the JVM. The dependency diagram shows specific implementations for these components, but in principal these components should be easily replaceable by other implementations of the relevant standards. Since the license conditions of each implementation may vary, our discussion of licensing issues will be based on the licenses of these particular implementations.

Some caveats may be in order. The argumentation toolbox as it has been presented here is only for the purpose of illustrating some Open Source license compatibility issues. We do not claim that the description of the system is complete or that all dependencies or their relevant licenses have been made explicit. And keep in mind that the argumentation toolbox is likely to change significantly during the course of the project, which will not begin until 2010.

Notice that four different Open Source licenses are used: the Gnu General Public License (GPL), the Eclipse Public License (EPL), the Apache License, and the PHP License.[1] Moreover, one component, the JavaFX runtime, currently uses a proprietary license from Sun Microsystems. The GPL is the license used most frequently, by 9 of the 14 components used by the toolbox, but it is not quite clear whether all of these components use the same version of the GPL. This could be discovered with further research. The Firefox web browser allows the license to choose between the GPL, the Lesser General Public License (LGPL) and the Mozilla Public License (MPL).

[1]Moreover, there is a risk that another license may apply to some subcomponent of a component, which might have been overlooked by the licensor of the component.

3 License Concepts and Issues

Let us now try to use the above scenario to obtain a better understanding of some basic software licensing concepts and issues. The *license* of each component grants exclusive intellectual property rights of the *licensor*, the *owner* of the software, to the *licensee*, subject to the conditions of the license. The intellectual property rights regulated by copyright law include the rights to *copy*, *modify* and *distribute* the original work owned by the licensor. (The owner need not be the author of the work, since copyrights can be assigned to others.) The *works* protectable by copyright are the "original works of authorship fixed in any tangible medium of expression, from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device." (17 U.S.C. § 102)[2] In the case of software, the work includes both the source code and the object code, as well as any documentation. Not covered by copyright are the algorithms or other ideas embodied by the software. Ideas are protected, if at all, only by patents. Copyright protects the expression of ideas in a perceivable medium. Copyright does however extend to cover translations. Thus, the translation of a computer program into another programming language is presumably a protected act of copying requiring permission of the owner of the copyright.

A modification of an original work is called a *derivative work*. The modifications of the derivative work are owned by the person who made the modifications. Let us assume the author of the modifications is a licensee of the original work. (Otherwise the modifications would have been an illegal infringement of the copyright of the original author.) The author of the modifications may license the modifications, since he owns them, to third parties. In this new license, the original work the author of the modifications will be the licensor and the third party will be a licensee. A sequence of modifications to a work, by several authors, creates a *chain of title*, where each author in the chain owns part of the work resulting at the end of the chain. Informally, we speak of moving 'upstream' and 'downstream' along a chain of title, as we move temporally backwards and forward, respectively, along the sequence of modifications to the work. As shown in Figure 1, modifying several prior original works can result in multiple, branching chains of title, forming a tree or, more generally, a directed network of derivations of works.

While the author of a derivative work is the owner of his modifications to the original work, the license of the original work may be subject to conditions which restrict the terms and conditions of licenses of these modifications to third parties. In particular, some Open Source software licenses, called *reciprocal licenses*, such as the GPL, require the modifications to be licensed under the same terms and conditions as the license of the original work. Actually this reciprocity requirement is even stronger. Not only must the terms and conditions of the license have the same meaning, reciprocal licenses require further that the modifications to be licensed using another instance of the same license template, using the exact same language as the license of the original work. (The distinction between a *license template* and a *license*, which can be an instance of such a template, is discussed below.) Reciprocal licenses are sometimes called *copyleft* licenses, which is a play on words emphasizing the supposedly leftist political

[2]Section 102 of Title 17 of the United States Copyright Act

agenda of the Free Software Foundation, which authored the GPL. Critics of this agenda have their own rhetoric and like to call reciprocal licenses “viruses” which “infect” software which is derived from Open Source software using reciprocal licenses. But whatever one’s political preferences may be, in its important to keep in mind that reciprocal licenses are just ordinary copyright licenses which copyright owner can chose to exercise their exclusive rights to control the copying, distribution, and modification of their intellectual property. Reciprocal licenses are in no way subversive of copyright law, but rather are a means of exercising rights protected by copyright law. Open Source licenses which do not have this reciprocity condition are called *academic licenses*, because one of the first and most popular academic license, the Berkeley Software Distribution license (BSD), was written by an academic institution, the University of California, Berkeley.

Whether or not the author of the modifications may license the entire derivative work to third parties, rather than just his own modifications, without the original work, depends on whether the initial license gives him the right to sublicense the original work. If not, third parties will have to obtain two licenses, one from the author of the original work and one from the author of the modifications. This is not typically a problem for Open Source licenses, since all the necessary licenses can be distributed along with the derivative work. Some Open Source licenses, including the BSD and Apache licenses, are not sublicensable. The leading Open Source license, the GPL, is not clear about sublicensing. A license which allows sublicensing will typically place conditions on the terms of the sublicense, usually requiring the terms and conditions of the sublicense to be the same as the original license.

The term ‘license’ is often used ambiguously to mean both the terms and conditions of a license template, such as the GPL, as well as the particular license granted by a particular licensor to a particular licensee for a particular work. Thus, strictly speaking, even when the same template license is used for a sublicense, such as the GPL, there are two licenses involved, one between the licensor and licensee of the first license, and one between the licensor and licensee of the second license, where the licensee of the first licensee is the licensor of the sublicense. This state of affairs is illustrated in Figure 2. When some license for a component of a collective or derivative work is not sublicensable, a template license distributed along with the derivative or collective work may in fact represent a number of instances of the license, with the owner of each component subject to a license which is not sublicensable remaining the licensor of the component with respect to downstream licensees.

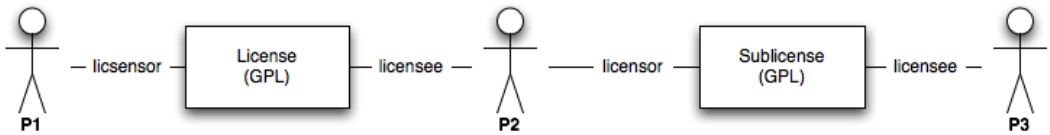

Figure 2: A Sublicense

The argumentation toolbox example illustrates two legal issues faced by the developers of Open Source software:

- What license options are available for each component and for the toolbox as a whole?
- Since components used by the system are subject to several different licenses, how can all of the required components be distributed together, or otherwise made available to users, without violating the terms and conditions of the licenses, in particular the 'copyleft' condition of reciprocal licenses such as the GPL?

Let's take a deeper look at the policy modelling component. It depends on the Clojure compiler, which uses the Eclipse Public License (EPL), and the Google Web Toolkit (GWT), which uses version 2 of the Apache License. These components in turn both make use of the Java Virtual Machine (JVM), licensed using the GPL, and the some operating system. Let's assume Linux is used, which of course also license using the GPL. The EPL and the GPL are both reciprocal licenses. If the policy modelling component is a derivative work of both Clojure and the JVM, then the policy modelling component would have to be licensed using both of these licenses, which may not possible if their reciprocity conditions are interpreted literally, since they both require the same license, the EPL and the GPL respectively, to be applied to derivative works. If this is indeed required, could this requirement be met by giving licensees of the policy modelling tool an opportunity to choose between the EPL and GPL, that by using a *dual license* scheme? Or are the reciprocity conditions stricter, requiring the software to be published using the same license, while offering no alternative license?

If the policy modelling tool is a derivate work, might the reciprocity condition of the GPL be satisfied by publishing the tool using the EPL, or vice versa? Although the GPL and EPL are not identical licenses, are their terms and conditions perhaps sufficiently compatible so as to be able to satisfy the reciprocity conditions of either license? The EPL is somewhat less restrictive than the GPL, in that it allows the object code of the modifications to the original work to be licensed under a proprietary license, i.e. one which is not Open Source, so long as the source code of the modifications, if published, is licensed using the EPL. This would seem to be a significant relaxation of the terms and conditions of the GPL, and speaks against the EPL being construed as a license which is compatible with the GPL. But even if the terms and conditions of the EPL were full equivalent in meaning with the GPL, both the EPL and the GPL are quite clear about requiring the source code of modifications to be published using the exact same license, not simply one with the same meaning. The EPL states that "a copy of this Agreement must be included with each copy of the Program", where the "Program" is the derived work. And Version 3 of the GPL states that the derived work "must carry prominent notices that it is released under this license" and requires the "entire work", including the modifications, to be licensed "under this License to anyone who comes into possession of a copy".

Can a license legally restrict the terms and conditions under which modifications owned not be the licensor but rather the licensee are published? To analyse this question, it may be necessary, depending on the governing law of the jurisdiction, to distinguish between *bare licenses* and licenses which are also *contracts*.[3] A bare license gives the licensee permission to exercise the exclusive rights of the

[3]In some jurisdictions, such as France, all licenses are contracts and the concept of a bare license is not recognized.

licensor. In the case of copyrights, these include the rights to copy, distribute and modify the original work owned by the licensor. A license can place conditions on a licensee when exercising the rights of the licensor, but can a bare license restrict the right of the licensee to exercise his own exclusive rights to the original works he, the licensee, creates? With a bare license, only the licensor makes promises, namely to allow the licensee to exercise the exclusive rights of the licensor under certain conditions. But the licensee of a bare license makes no promises regarding his own rights. The situation is different if the license is a contract. A contract is an agreement in which the parties make promises to each other, where rights are transferred in both directions. Contract law is very liberal regarding the kinds of promises which may be made between the parties, and there seems no reason to suppose a promise by the licensee to publish any derivative works under the same license would not be enforceable. However, in order for a license to become a binding contract, certain conditions have to met, which vary from jurisdiction to jurisdiction. In Common Law countries, such as the UK and the US, a license contract typically requires an *offer* by the licensor to be *accepted* by the licensee, and for something of value, called *consideration* to be mutually exchanged in both directions between the parties. It can be problematical to prove that all three of these conditions have been met in particular cases. If an Open Source program is distributed via the Internet, some means of assuring that people who download the software are made aware of the terms and conditions of the offer and have to do something to express their subjective acceptance of these terms and conditions, may be necessary. This is the reason that when downloading software on the Internet, the license is often first displayed and one is required to click some box to explicitly express acceptance of the license, before the software can be downloaded. But what if the licensee obtained the software from some other source, for example from a friend or colleague? Just as problematic is the question of consideration. Since Open Source software is typically offered free of charge, what consideration flows from the licensee to the licensor? Would a promise to publish derivative works under the same license be sufficient consideration?

All of these issues are relevant only for derivative works. But is the policy modelling tool a derivative work of either of the Clojure compiler or the JVM? Clojure is a compiler for a new programming language for the JVM. Part of the policy modelling tool will be written in the Clojure language. But the bytecodes produced by the compiler are plain JVM bytecodes, exactly as if the program had been written in Java. For the sake of argument, let us assume that the Clojure compiler is not used at runtime, but only when compiling the source code of the policy modelling tool to create an executable program for the JVM. Thus the compiler, which is the component subject to the EPL, need not, in principal, be distributed with program. However the program also makes use of the Clojure runtime and libraries. It would be very inconvenient for users if the runtime and libraries could not be distributed with the program and needed to be downloaded separately. These libraries and runtime are also intellectual property subject to copyright law, and are also licensed using the EPL. Thus, running the argument modelling tool requires libraries licensed under the EPL to be linked with the JVM runtime licensed under the GPL. Can this be done without violating the reciprocity conditions of both licenses? The Free Software Foundation, which authored the GPL, claims that linking a program to a library licensed by the GPL requires the

reciprocity condition of the GPL to be applied to the program, meaning that, in their view, such a program must also be licensed under the GPL. Would it make a difference if the policy modelling tool is not distributed together with the Clojure runtime and libraries or the JVM? What if the user of the argumentation tool would have to download and install the these components separately from the policy modelling tool and would link them himself when running the program? What if the source or object code of the policy modelling tool is only made available only as part of some kind of network service, such as a web service, so that the program is used by sending messages to the tools over the network, perhaps using a custom protocol? Would providing such a web service count as a publication or distribution which requires the policy modelling tool, as a derived work, to be published as Open Source software, since it uses, on the server, GPL licensed software, such as Linux and the JVM?

But is the standpoint of the Free Software Foundation regarding linking legally correct? Does merely linking a program to a library create a derivative work, or would distributing the program together with the libraries only result in a *collective work*, not subject to reciprocity condition? After all, neither the source code nor the object code of library library is modified by linking other code to it, anymore than including two separate programs on one disk modifies these programs, or linking one web page to another, via a URL, modifies the referenced page. If a web page is licensed using a reciprocal license, would any page which linked to it also be subject to the same license? To make another analogy, imagine publishing a scientific article via print media using a reciprocal license, such as one of the reciprocal Creative Commons licenses[4]. Can you imagine another scientific article which 'links' to the first via a reference or citation being subject to the reciprocity condition and also having to use the same license? Whatever the position o the Free Software Foundation regarding how to interpret the GPL, it is the courts alone who have the legal authority to decide whether or not a particular use of licensed software creates a derivative work. And courts will decide this issue on a case-by-case basis against the background of their own legal system.

Some Open Source licenses, but not the GPL, include clauses which aim to limit the *jurisdiction*, *venue*, or *governing law* for resolving disputes concerning the license [8, p 218]. Jurisdiction determines which courts have the power to the decide the case, such as US federal courts. The venue determines the particular location of the court, such as the US District Court of the Southern District of California, in San Diego, California. And the governing law determines which law shall be applied to decide the legal issues of the case. Usually the governing law will be the law of the jurisdiction of the court which decide the case, but this is not always the case. Sometimes courts need to be apply the law of other jurisdictions, possible even the law of other countries. For example, the Open Software License (OSL) allows an action to be brought in the courts of the jurisdiction where the licensor resides, wherever this may be, but requires US copyright law, "the equivalent laws of other countries" and international treaties to be applied to determine penalties.

What about the dependency of the policy modelling tool on the Google Web Toolkit (GWT)? The GWT is a library for the JVM for creating interactive web applications using the asynchronous JavaScript and XML programming paradigm

[4]http://creativecommons.org/about/licenses/meet-the-licenses

(AJAX). The GWT allows the graphical user interface of a web application to be implemented in Java, by compiling Java to JavaScript. At runtime, static HTML pages containing JavaScript code are distributed from the web server hosting the application to the user's web browser. The GWT library is used only during the development of the user interface, not at runtime. Is the JavaScript code generated by the GWT library subject to the same license as the GWT library itself? Although, the GWT library is licensed using the Apache license, which is an academic license without a reciprocity condition, the Apache Software Foundation has started to require *contributors*, i.e. persons who modify code licensed under the Apache license, to enter an agreement which requires the author of the modifications to enter an agreement with the Apache Software Foundation, giving the Foundation the right to republish the modifications under any license it chooses [8, p 93]. Is every program derived from code licensed using the Apache template license a "contribution" subject to this condition? Does using the GWT library to implement a user interface create a derivative work, modification or contribution requiring the author of the program to make such an agreement with the Apache Software Foundation or perhaps Google, the author and owner of the GWT?

The legal issues raised by the other components of the argumentation toolbox are similar. Interesting issues may be raised by the use of JavaFX by the argument mapping tool, since the JavaFX runtime currently is licensed by Sun using a proprietary licensed, as closed-source software. (The JavaFX compiler, however, is licensed as Open Source software using the GPL, version 2.) The proprietary license does not grant the right to distribute the JavaFx runtime. In practice this is not likely to be a problem, since JavaFX programs can be distributed as applets and Java WebStart applications, both of which are capable of downloading and installing the required Java and JavaFx runtime environments from Sun's own servers automatically, on demand, the first time the user tries to run the program. Since the JavaFX runtime is written in Java and runs on the JVM, one question is whether Sun may be violating the GNU license of the its own JVM. But since Sun is the owner of its own JVM, and the licensor of its GPL license, they are not themselves bound by its GPL license conditions. They are free to use their own JVM as the please, unconstrained by any licensing conditions. Moreover, since the JVM has a standard specification, JSR 924[5], which has been implemented many times, software written for the JVM is not dependent on any particular JVM implementation. The owner of each JVM implementation is free to choose its licensing conditions, including proprietary, closed source ones.

[5]http://en.wikipedia.org/wiki/JVM

4 Sources of Law

In the previous section, many potential Open Source licensing issues were illustrated, using the argumentation toolbox example. We did not try to give a definitive answer to any of these legal questions for several reasons. Such questions can only be answered in the context of a particular legal jurisdiction, and answering them would require advice from an attorney who is expert in the law of this jurisdiction. Whatever these answers might be, they would be of only limited relevance to readers in other jurisdictions. Thus our goal has been more limited: to provide a high-level conceptual model of Open Source licensing issues and some more-or-less concrete examples of the kinds of issues which may need to be addressed by developers of Open Source software. The conceptual model is based on US copyright law, since the most widely-used Open Source licenses were written in the US and are thus informed by and based on US copyright law and its concepts and terminology. When analysing Open Source licensing issues, however, one must dig deeper and cannot interpret these licenses literally or evaluate them independently of the legal tradition, positive law and jurisprudence of the governing law of specific cases. In this section we provide an overview of the sources of legal norms which should be taken into consideration when analysing Open Source licensing issues, starting with but going beyond the literal text of the licenses themselves.

Licenses and contracts provide a means for private individuals and companies to regulate their own business affairs. Licenses are like little pieces of legislation, which regulate the distribution of certain rights and obligations between the licensor and licensee regarding original works of software owned by the licensor. Thus the first source of legal norms for analysing licensing issues are the licenses themselves. But Open Source licenses, like legislation, change over time. There are different versions of popular Open Sources licenses in use. For example, the current version of the GPL is version 3. When several versions of a license have been published, it may not also be apparent which version applies to a particular piece of software. The license itself may provide licensees with the option to use newer versions of the license for derivative works. For example, section 14 of version 3 of the GPL states:

> Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

But what if a work is derived from several components, each subject to different versions of the GPL? Recall that works licensed using the GPL are not sublicensable. Thus one may need to examine all programs along the chain of title to find out, for each modification, whether the owner of the modification intended to allow his derivative work to be licensed by a specific version of the GPL or any later version. This also raises the issue of which version of the GPL must be applied to software which is derived from more than one program, when

these programs use different, specific versions of the GPL, in order to satisfy the reciprocity conditions of each of the GPL licenses.

Again, it is not sufficient to interpret licenses literally. They must be interpreted against the background of the legal tradition, positive law and jurisprudence of the law governing the license. Usually this will be the law of the jurisdiction in which a case would be brought before a court to resolve legal conflicts concerning the license. For example, the reciprocity condition of Version 3 of the GPL is stated in its Section 5, as follows:

> 5. Conveying Modified Source Versions. You may convey a work *based on the Program*, or the *modifications* to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions: ... The work must carry prominent notices stating that it is released under this License ... You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. ... (Emphasis added.)

So, on the face of it, the reciprocity condition of the GPL applies both to works "based on" the licensed software and to works which result from "modifications" to the licensed software. But the definitions section of the GPL makes it clear that a work is based on an earlier work only if it is a modified version of this earlier work:

> To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

But "modified version" is not a legal term in US copyright law. Rather, US copyright law uses the term "derivative work", which is defined in 17 U.S.C. § 101 as follows:

> A "derivative work" is a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a "derivative work".

So, when US copyright law is the governing law, it may be necessary to interpret the term "modified version" in the GPL in terms of the technical legal term, "derivative work", of 17 U.S.C. § 101. US copyright law gives the owner the copyright the exclusive right to make derivative works, but unlike the GPL says nothing explicitly about the making of modifications or works "based" on other works. A license gives the licensee permission to exercise an exclusive right of the licensor. If not all modifications are not derivative works, then those modifications which are not derivative works arguably do not infringe on the licensor's exclusive rights and would not require a license. What then is the precise relationship between "modified version" and "derivative work". Are they equivalent? Is one term more general than the other, i.e. do one term *subsume* the other? If they are not equivalent, is the reciprocity condition of the GPL binding on modified versions which are not derivative works?

Does it make a difference whether the GPL is interpreted as a bare license or a contract? If the GPL is interpreted as a bare license under US copyright law, arguably it can only places conditions on the exercise by the licensee of exclusive rights owned by the licensor. According to 17 U.S.C. § 106, these exclusive rights are:

> ... the owner of copyright under this title has the exclusive rights to do and to authorize any of the following: (1) to reproduce the copyrighted work in copies or phonorecords; (2) to prepare derivative works based upon the copyrighted work; (3) to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease, or lending; ...

The rights to perform and display works have been omitted from this quotation, since these rights do not apply to software. Arguably, if the GPL is a bare license, any conditions it places on actions which fall outside the exclusive rights of the licensor would not be binding on the licensee. In particular, if the GPL's concept of "modifying" a work is broader than the legal meaning of "preparing derivative works" under copyright law, then the GPL's reciprocity conditions would only be binding for those modifications which are derivative works. Similarly, if the definition of "conveying" a work in the GPL is broader than concept of "distributing copies" of copyrighted works in copyright law, then the reciprocity condition of the GPL would only apply to conveyances which are distributions.

Consider also the linking issue. At the end of Version 3 of the GPL, but significantly after its terms and conditions, in the section on "How to Apply These Terms to Your New Programs", there appears this paragraph:

> The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. ...

This is the only place in the GPL where linking is explicitly mentioned. But since it appears after the terms and conditions of the license, it is not part of the legal binding conditions of the license. Moreover, even if these paragraph were part of the terms and conditions of the GPL, it does not tell us what it means to "incorporate a program into proprietary programs" or "link proprietary applications with a library". If these actions do not prepare derivative works or distribute copies of the copyrighted work, then they fall outside the exclusive rights protected by copyright and no bare license. This raises the issue whether a bare license can place conditions prohibiting actions which the licensee has a prior right to perform, which do not require permission of the licensor. Can the conditions go beyond limitations on the exercise of the exclusive rights of the licensor?

The situation may be different if the license is a contract and not just a bare license. In general, the parties to a contract are free to make promises which go beyond respecting conditions in return for permission to exercise exclusive rights. The Free Software Foundation intends the GPL to be a bare license, not a contract [7, pp 136-140]. Indeed, the GPL is the only popular Open Source license which is not intended to be a contract [8, p 140]. But is the intent of the Free

Software Foundation relevant? Or is it rather the intent of the licensor and licensee which counts when determining whether or not a contract has been formed? Let us leave aside the question of whether or not the GPL can be a contract. The important point for us here is that Open Source licenses are often intended to be contracts by the parties, and most license templates have been designed to be used as contracts.

If a party in a copyright dispute claims that a license is a contract, then contract law, in addition to copyright law, becomes relevant. In the United States, copyright law is federal law but contract law is state law. That is, the law of contracts is somewhat different in each of the 50 states. But the situation is not as complex as it may seem at first. The American Law Institute has published a jurisprudential treatise called the "Restatement of Contracts" [5], which presents an abstract model of US contract law, derived from an analysis of court decisions in contract disputes throughout the United States. The model is presented in the form of legislation, that is as a set of rules for the law of contracts. Although the Restatement has the form of legislation, it is important to keep in mind that it is not a *primary* source of law, such as legislation or court decisions, but rather a *secondary* source of law, written by academic lawyers expert in the field of contract law.[6] Although the Restatement of Contracts is a very useful reference, helpful for getting an overview of the US law of contracts, it is no replacement for the primary legal sources, the statutes and court decisions of the governing law of the relevant jurisdiction.

Also relevant in the US is the Uniform Commercial Code [11]. The Uniform Commercial Code (UCC) is a federal law which aimed to harmonize the law of commercial contracts in the 50 US states. But since contract law is state law in the US, the UCC is not binding on the states but rather only serves as guidance for the state legislatures. Some version of the UCC has been enacted by all 50 states. But the case law interpreting the UCC in each state, or rather the particular version of the UCC enacted in each state, can in principle diverge. The decision of a court in one state is not binding on decisions by courts in other states, but can be influential. Thus the UCC is similar in some ways to a directive of the European Union, which directs member states of the EU to enact legislation implementing the directive. There is no guarantee, despite good faith efforts, that national laws implementing the directive will have exactly the same meaning in each EU member state. One possible difference between an EU directive and a US uniform law is that the US states have no formal legal obligation to implement the uniform law.

Whether or not the UCC is relevant for resolving Open Source licensing issues will likely depend not only on whether the license is a contract, but whether the contract is for the sale of goods. Thus important issue is whether software is or can be a "good". UCC § 2-103(k) defines goods as follows:

> (k) "Goods" means all things that are movable at the time of identification to a contract for sale. The term includes future goods, specially manufactured goods, the unborn young of animals, growing crops, and other identified things attached to realty as described in Section 2-107. The term does not include information, the money in which the price is to be paid, investment

[6]In Germany, this kind of jurisprudential research is conducted in the field called "Rechtsdogmatik".

securities under Article 8, the subject matter of foreign exchange transactions, or choses in action.

Is software a "movable thing"? Commonsense might seem to tell us that it is. But if you buy Microsoft Word, for example, do you really buy the software, per se, or rather a license giving you permission to use the software? Clearly Microsoft retains its ownership of the software. It does not assign its ownership of intellectual property rights to you when you buy a copy, otherwise Microsoft would not be free to continue selling Word to others. Thus the question of whether software is a good becomes whether a license to exercise some exclusive right, a copyright, can be a good? Both software and licenses are quite intangible objects compared to the kinds of things we usually consider to be goods, such as automobiles, computers or television sets.

Another issue which must be resolved to determine whether the UCC applies, besides whether software is a good, is whether the license was issued as part of a commercial transaction. Typically, Open Source software is distributed free of charge, but collections of Open Source software on some medium, such as a CD or DVD, are often offered for sale. Are the licenses for the software in the collection part of the commercial transaction, or does the commercial part of the transaction only cover the sale of the medium, the CD or DVD?

We have been focusing in our discussion thus far on US law. In Europe of course Open Source licensing issues typically would be resolved according to European law. The European Union now has 27 member states, each with their own national laws, including both common law countries (Cyprus, Ireland, Malta and the United Kingdom) and civil law countries. Moreover the EU has 23 official languages, which exacerbates the problem of interpreting Open Source licenses, which are typically instances of license templates, such as the GPL, written in English. The Free Software Foundation publishes translations of the GPL in several languages, but these translations are considered "unofficial".[7] This diversity of copyright laws is mitigated to some extent by international treaties and efforts of the European Union to harmonize the copyright law of member states. All member states ratified the 1886 Berne Convention for the Protection of Literary and Artistic Works.[8] Prior to the Berne Convention, national copyright laws usually only applied to works authored within the country. Works authored in other countries could be freely copied, modified or distributed without permission. The protection of the interests of authors provided by the Berne Convention went beyond the protection of economic interests provided by US copyright law at that time to also cover the "moral rights" of authors.[9]

More recently, the European Union has issued a number of directives to its member states in an attempt to harmonize copyright laws throughout the EU.[10] Of particular interest for software licenses is Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs.[11] This directive protects the same basic exclusive rights of copyright owners to copy, modify and distribute

[7]http://www.gnu.org/licenses/translations.html

[8]http://en.wikipedia.org/wiki/Berne_Convention_for_the_Protection_of_Literary_and_Artistic_Works

[9]The United States also ratified the convention, but not until 1989!

[10]http://en.wikipedia.org/wiki/Copyright_law_of_the_European_Union

software, similar to US copyright law, but defines some exceptions which limit these rights in some situations. Licensees have the right to make any copies necessary to use the software, the right to make any modifications necessary to function properly, given its purpose, for example by fixing bugs, the right to make back-up copies and, finally, the right to decompile the program if necessary to ensure that the program is interoperable with other programs or devices. The important thing to note here is that this European directive gives licensees these rights even if the license itself does not grant these rights or, presumably, even if the license explicitly tries to deny the licensee these rights. Council Directive 91/250/EEC has been implemented in most, perhaps all, member states of the European Union, including France[12] and Germany [13]

The provisions of the European directive are reminiscent of the consumer protection provisions of the Uniform Commercial Code. For example, UCC § 2-314 states an implied *warranty of merchantability* for sales of goods by merchants which essentially guarantees the buyer that the goods purchased are fit for the purposes for which they were designed. In such cases, any clause of the sales contract which attempts to exclude a warranty of merchantability is unenforceable.

The EU directive on the legal protection of computer programs and the UCC are both examples of laws which can override the express conditions of a license or contract. This illustrates that one cannot rely solely on the literal text of a license or contract to determine the rights and obligations of the parties.

Council Directive 91/250/EEC was replaced this year by Directive 2009/24/EC to consolidate minor amendments over the years.[14] Just as template licenses like the GPL can change over time, raising issues about which version of the license applies to a derivative work, so to do statutes and case law evolve over time. When analysing a legal case, timing issues can be difficult. The general rule is that the law which was valid at the time of the events which give rise to the facts of the case must be applied, not the law which is valid at the time of the court proceedings to resolve the conflict. But these events can take place over any period of time and in principal the law may have changed, perhaps even several times, during the course of these events. In the case of new legislation, one must distinguish the time which the statute was enacted by the legislature from the time which the statute becomes effective and the period of time for which the statute is applicable. Some laws are applicable retroactively. Some are applicable only for some period of time in the future. Court interpretations of legislation apply retroactively on the theory that the courts do not make law, but only interpret the meaning of legislation already valid at the time in the past of the relevant events of the case before the court. But since courts can interpret the language of statutes in ways which surprise expectations, such interpretations may seem at times to be in effect a change in the law.

Let us conclude this section by mentioning only briefly some general sources of law, not specific to copyright issues, which nonetheless may be relevant in some cases and therefore must be considered. The first is constitutional law, i.e. the

[11]http://en.wikipedia.org/wiki/Directive_on_the_legal_protection_of_computer_programs

[12]Loi no. 94-361 du 10 mai 1994, JORF du 11 mai 1994, p. 6863

[13]Zweites Gesetz zur Änderung des Urheberrechtsgesetzes vom 9. Juni 1993, BGBl I p. 910

[14]http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:111:0016:0022:EN:PDF

basic law of every jurisdiction. Which constitutions need to be considered depends of course on the jurisdiction. Several may be relevant. Recall that contract law is state law in the US. Thus, if the governing law of the case, for contract issues, is the law of California, both the California State Constitution and the US Constitution may be relevant. Similarly, for a case in Germany, the constitution of the particular Germany Land, for example Brandenburg, as well as the German constitution may be relevant.[15]

The European Union does not currently have a constitution, since the treaty of 2004 which produced a proposed constitution was not ratified by all 25 member states. In the meantime, several European treaties need to be consulted, in particular the Maastricht Treaty of 1993, which founded the European Union, and the Lisbon Treaty of 2007, which amended all previous treaties after the proposed constitution of 2004 failed to be ratified. The highest court with jurisdiction for interpreting these treaties and deciding issues of European law is the European Court of Justice, which was established in 1952.

Possibly also relevant is the European Convention on Human Rights, under the auspices of the Council of Europe in 1950. The Convention also established the European Court of Human Rights to protect persons from human rights violations. The Convention is comparable to the US Bill of Rights, i.e. the first ten amendments of the US constitution, the French Declaration of the Rights of Man and the first part of the German Basic Law. Possibly of particular relevance for copyright issues is Article 10 of the Convention, which protects the right of "expression":

> Article 10 – Freedom of expression
>
> 1. Everyone has the right to freedom of expression. This right shall include freedom to hold opinions and to receive and impart information and ideas without interference by public authority and regardless of frontiers. This article shall not prevent States from requiring the licensing of broadcasting, television or cinema enterprises.
>
> 2. The exercise of these freedoms, since it carries with it duties and responsibilities, may be subject to such formalities, conditions, restrictions or penalties as are prescribed by law and are necessary in a democratic society, in the interests of national security, territorial integrity or public safety, for the prevention of disorder or crime, for the protection of health or morals, for the protection of the reputation or rights of others, for preventing the disclosure of information received in confidence, or for maintaining the authority and impartiality of the judiciary.

Constitutions, treaties and conventions such as these generally state very broad principles, not detailed rules with well-defined terms. Other sources of principles may be recognized by courts, depending on the legal system. For example, Common Law countries recognize principles of *equity*, such as the principle of *estoppel* which, roughly speaking, expresses the idea that a person should not profit from his own wrongdoing. Principles can play a role when courts decide cases, even though they are not codified in statutes enacted by the legislative

[15]For historical reasons, the constitution in Germany is not called a constitution, but rather the Basic Law ("Grundgesetz").

branch of government. One issue is whether the enforcement of moral principles, outside of positive law, is consistent with the liberal ideals of western democracies. Are principals of *natural law* fundamental and universal and should they be given priority over the positive laws of states?

5 Legal Reasoning and Argumentation

Now that we have identified and illustrated some Open Source license compatibility issues, and collected some sources of legal norms that may be relevant for resolving such issues in particular cases, let us turn in this section to the question of how to use such legal sources to analyze the issues. Without diving too deeply into legal theory, our aim here is to present a very high level and brief overview of legal reasoning and argumentation, from the perspective of the interdisciplinary field of Artificial Intelligence and Law. Research in this field builds on results from both jurisprudence and computer science and pursues the goal of building computational models of legal reasoning, for both theoretical and practical purposes. In a later Qualipso technical report, models, methods and tools developed in the AI and Law community will be presented and assessed with regard to their applicability for providing support to developers with the kinds of Open Source license compatibility issues. Here our aim is to present an overview of legal reasoning tasks and relationships between these tasks, without saying much about how these tasks could be supported with information technology.

Legal positivists, such as Hart [4], take the position that governing law of a jurisdiction consists of a set of legal rules. Rules are of two kinds. The *primary rules* represent the legal norms which regulate the legal relationships and activity of citizens and other persons. The *secondary* rules represent procedural norms which regulate the processes by which legislatures and courts construct, modify and apply the primary legal rules. More concretely, secondary rules govern the process by which courts identify the primary rules and apply them to decide legal issues in particular cases. A simplistic conception of this process, not Hart's, disparagingly called *mechanical jurisprudence*, trivializes the task of identifying the legal rules, as well as the facts of a case, and considers the application of rules to cases to decided issues a straightforward, unproblematical application of deductive logic. Mechanical jurisprudence fails to recognize or take seriously the difficulties of interpreting legislation which are written in natural language, with all its potential vagueness, ambiguities and imprecision. As Dworkin [2] however has pointed out, legal sources such as legislation and case law need to be *interpreted* to identify the legal norms, and interpretation is anything but a straightforward, mechanical process. And many legal concepts are *open textured* [4] abstract concepts which must be interpreted against the background of such things as the legislative history, precedent cases and values of the relevant community when trying to decide whether concrete, material facts of a particular case can be *subsumed* under them. To illustrate with an example from copyright law: the owner of a copyrighted work has an exclusive right to *distribute* copies of the work. But what does "distribute" mean precisely? Is a copy of a program being distributed when it is shared with friends or immediate family members? It is not always clear how to answer such questions.

The mainstream view within the field of AI and Law is that legal reasoning involves the construction, evaluation and comparison of alternatives theories of the law and facts of the case. Typically this takes place in critical dialogues, during which arguments pro and con the alternative theories are put forward by the parties. When evaluating and comparing the alternative theories, the most *coherent* ones should be preferred, but just what it means for a theory to be coherent is an open

theoretical issue. Some factors which are relevant for evaluating the coherence of a theory have been put suggested, but there are not strict rules or formulas for aggregating these factors. For example, one factor is how well the theory fits into the prior body of case law. Another factor is the complexity of the theory. Following the principle called Occam's razor, simple theories are preferable to more complex theories.

In this theory construction conception of legal reasoning, two or more theories can be equally coherent. These theories can point in different directions, leading to contradictory conclusions of the issues. Further arguments, for example from new evidence or interpretations, may resolve these conflicts and lead to a unique, best theory. But this still provides not guarantee a decision would be definitely correct if taken at this time, on the basis of this theory. Given more time, if the dialogue is allowed to continue, still further arguments might be found and put forward which lead to still better theories. Thus, even if one accepts the idea that every legal issue has one, definitely correct answer, in principal at least, in practice legal procedures are imperfect. There is no objective method, independent of these imperfect legal procedures, for checking whether a legal decision is correct. Rather, in practice we have to live with some doubt and must be content with merely presuming legal decisions to be correct, at least so long as there is no evidence to the contrary.

The theory construction view of legal reasoning, in various forms, has been hinted at numerous times. For example, Rawls [7] said:

> General moral principals and judgments about the morality of specific acts are constructed together, in an iterative process of mutual adaptation.

In German jurisprudence, Engisch [3] famously said:

> One's attention must shift back and forth ("Hin- und Herwandern des Blickes") between the evidence and legal sources when trying to subsume facts under legal terms.

One of the founders of the field of computer science and law, Bing [1], wrote:

> Legal reasoning is not primarily deductive, but rather a modelling process of shaping an understanding of the facts, based on evidence, and an interpretation of the legal sources, to construct a theory for some legal conclusion.

Finally, this theory construction conception of legal reasoning was made very explicit in the field of AI and Law by, among others, McCarty [6]:

> Legal reasoning is a form of theory construction. … A judge rendering a decision is constructing a theory of [the law and facts of] a case. … A lawyer's job is to construct a theory of the case too, and one that just happens to coincide with his client's interests.

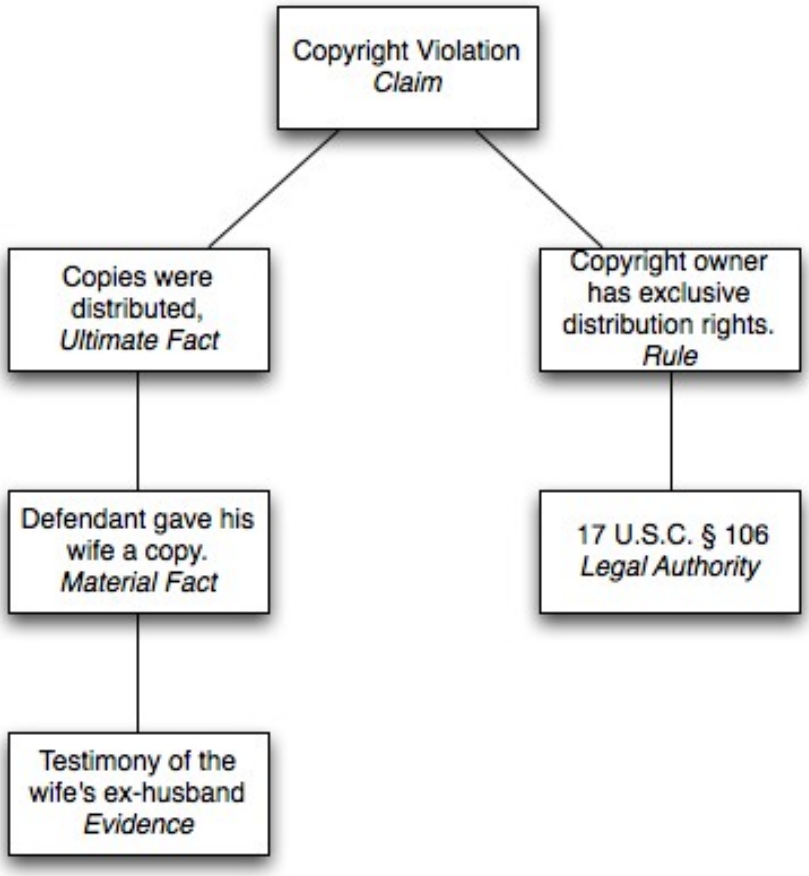

Figure 3: Kinds of Legal and Factual Issues

Figure 3 illustrates relations between different kinds of legal and factual issues, all of which are resolved by argumentation. The plaintiff's main claim is that the defendant violated his copyright by giving his wife a copy of some software. This claim is supported by an argument with two premises: the major premise, asserting the *rule* that copyright owners have the exclusive right to distribute their works, and the minor premise, expressing the antecedent of the rule, namely that the defendant in fact distributed a copyrighted work. The propositional content of the minor premise is called an *ultimate fact*, since it is expressed in the same terms, and at the same level of generality, as the antecedents of the legal rule being applied. That is, the ultimate facts are formulated using technical legal terminology. Putting forward this argument does not by itself resolve the main claim, that there was a copyright violation. On the contrary it raises two new issues which need in turn to be resolved by argumentation: 1) Is the asserted rule about distributing copies a valid legal rule? And 2) What did the defendant do, more concretely, that is claimed to be a distribution of copies? For the first of these issues, the claimed rule is *backed* [9] by putting forward an argument citing the source of legal statute, 17 U.S.C. § 106. The plaintiff is arguing that the claimed rule is a coherent interpretation of this section. Regarding the second issue, about what the plaintiff is claiming the defendant did, more concretely, which amounts to an illegal distribution of the copyrighted work, the plaintiff has put forward an argument claiming that the defendant gave his wife a copy. When

the propositional content of a claim is relatively concrete, using everyday terminology, rather than technical legal vocabulary, the proposition is called a *material fact*. Calling ultimate facts and material fact "facts" does not mean that they are undisputed or settled. In this context "fact" is a synonym for a proposition about factual issues, as opposed to legal issues, independent of whether or not the propositions are true, or presumably true. In our example, the claim of the material fact, that the defendant gave his wife a copy, is at issue. The plaintiff has supported this claim by putting forward yet another argument, this time by providing evidence in the form of witness testimony for the ex-husband of the defendant's wife.

Arguments are typically *enthymemes* [10]. Some of the premises of the argument are implicit. One way to attack an argument is to first reveal an implicit premise and then to put forward an argument against (con) the premise. For example, an implicit premise of the argument citing 17 U.S.C. § 106 is that this section of the U.S.C. is still valid law and has not been modified or repealed before the relevant events of the case. And an implicit premise of the argument from witness testimony is that the witness is not biased. The defendant might want to reveal this premise and challenge the witness in an argument which points out that an ex-husband may be jealous and thus have a motive to try to harm the defendant, who is the wife's new husband. For a third example, an implicit premise of the main argument is that the defendant did not have a license giving him permission to distribute the software. Thus the defendant might consider countering this argument by claiming that he has a license.

There are various ways to attack arguments: by attacking a premise, by putting forward an argument, called a *rebuttal* for a contrary conclusion or a conclusion, or by *undercutting* the argument with an argument claiming that the rule of its major premise does not apply in this case. For an example of an undercutter, imagine an argument applying an exclusionary rule stating that 17 U.S.C. § 106 does not apply to software, or to noncommercial distributions.

The process of making claims, putting forward arguments and deciding issues is regulated by rules of procedure. These procedural rules regulate, among other things, the distribution of the *burden of proof* among the parties and the *proof standard*, such as the civil law *preponderance of evidence* standard for resolving issues.

At some point in the proceeding, after all the evidence has been heard and all of the arguments have been made, the arguments will have to be evaluated. In legal trials, this is done by judges and, in some legal systems, juries. In the US, if there is a jury, the trial judge is responsible for deciding legal issues and the jury is responsible for deciding only factual issues. In theory, both the legal and the factual issues are resolved by evaluating the theories put forward by the arguments in the case and comparing their coherence. For the factual issues, one way of judging coherence is to evaluate which theory of the facts makes the plausible *story*, given common sense knowledge of how people normally behave and the world typically works. For the legal issues, the judge is not bound by the legal theories put forward by the parties but may choose to construct his own theory of the legislation and precedent cases and then decide which of the rules put forward by the parties are members of his preferred theory.

Thus far in this section we have been discussing how arguments are used in legal trials, to resolve a dispute. The scenario we presented in Section 2, about the Open Source license compatibility issues faced by the developers of an argumentation toolbox, is somewhat different. The scenario is an example of a legal *planning* problem, where the task is to anticipate the legal consequences of alternative courses of action, so as to try to *avoid* legal conflicts down the road. Nonetheless, argumentation plays a role. The planner needs to try to imagine potential legal issues and then search for arguments on both sides, simulating a dialogue by alternating between the plaintiff (pro) and defendant (con) roles and using arguments to construct and critically test theories for both sides.

6 Conclusion

In this report we have illustrated some license compatibility issues which developers must face when combining components subject to different licenses, using an argumentation toolbox currently being designed as an example, and have surveyed the kinds of legal sources, such as statutes, case law and legal principles, which must be taken into consideration when analysing these issues. We have seen that Open Source license compatibility issues cannot be analyzed in the abstract, but must be analyzed in the light of the particular material facts of a case and the legal norms of the applicable jurisdiction.

In the law, in practice there is never a uniquely right answer to some legal issue. Even if one takes the position that in principal there must be one right answer, in practice reasonable people can and will disagree about what this answer should be. Good arguments can always be made on both sides of any issue. Deciding legal issues requires good judgement, not just good logic. Legal problems are not well-formed and thus cannot be fully automated. Legal reasoning is a creative, synthetic process involving the construction, evaluation and comparison of theories. While formal, analytical methods can be useful for analysing the logical consequences of these theories, no formal method can generate all possible theories, since the search space of theories is not enumerable. This nature of legal reasoning leads to some necessary uncertainty and risk which cannot be entirely eliminated. This is as true for Open Source software development as for any other activity regulated by law.

In our next report we will survey methods from the field of Artificial Intelligence and Law to try assess their usefulness for building software tools which can help developers, or perhaps their attorneys, to construct and explore the space of legal arguments about Open Source license compatibility issues. The goal is not to build an intelligent system which can answer questions about Open Source license compatibility issues in a fully automatic way, but rather to develop tools which can help humans to analyse license compatibility issues more efficiently and more thoroughly.

References

[1] Jon Bing. Uncertainty, decisions and information systems. In C. Ciampi, editor, *Artificial Intelligence and Legal Information Systems*. North-Holland, 1982.

[2] Ronald Dworkin. *Taking Rights Seriously*. Harvard University Press, Cambridge, MA, 1977.

[3] K. Engisch. *Logische Studien zur Gesetzesanwendung*. C. Winter, 1960.

[4] H. L. A. Hart. *The Concept of Law*. Clarendon Press, Oxford, 1961.

[5] American Law Institute. *Restatement (Second) of Contracts*. 1981.

[6] L. Thorne McCarty. Some arguments about legal arguments. In *International Conference on Artificial Intelligence and Law*, pages 215–224, Melbourne, 1997.

[7] John Rawls. Outline of a decision procedure for ethics. *Philosophical Review*, pages 177–197, 1951.

[8] Lawrence E. Rosen. *Open source licensing: Software freedom and intellectual property law*. Prentice Hall Professional Technical Reference, Upper Saddle River, New Jersey, USA, 2004.

[9] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.

[10] Douglas Walton. *Fundamentals of Critical Argumentation*. Cambridge University Press, Cambridge, UK, 2006.

[11] James J. White and Robert S. Summers. *Handbook of the Law Under the Uniform Commercial Code*. West Publishing Co., 1980.