QualiPSo

Quality Platform for Open Source Software

IST- FP6-IP-034763

Deliverable A1.D2.1.3

Report on Prototype Decision Support System for OSS License Compatibility Issues

Thomas F. Gordon
Fraunhofer FOKUS, Berlin

Due date of deliverable: dd/mm/yyyy

Actual submission date: 23/9/2010

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

Change History

Version	Date	Status	Author (Partner)	Description
0.8	04.08.10		Tom Gordon (Fraunhofer)	initial draft
0.9	01.09.10		Benjamin Egret (INRIA)	corrections
1.0	23.09.10	final	Tom Gordon (Fraunhofer)	

Executive Summary

Building on Semantic Web technology and our prior theoretical and practical work on the Carneades argumentation system, we have developed a proof-of-concept, prototype system for helping developers to construct, explore and compare legal theories when analysing open source licensing issues in particular cases. The prototype takes into consideration an analysis of requirements. This analysis concludes that the resolution of open source licensing issues is an argumentative process in which alternative theories of copyright law concepts, such as the concept of a derivative work, together with the facts of particular cases, are constructed and critically evaluated. An ontology of open source licences has been developed, using the Web Ontology Language (OWL) and this ontology has been used to model several popular open source licenses, including the Apache 2.0, BSD and MIT academic licenses, as well as the MPL, EPL and GNU GPL reciprocal licenses. Several variants of the GNU GPL are included in the model, including the GNU AGPL and the GNU LGPL. In addition we have developed an ontology for describing software projects, including various relationships between software entities used by the project, at the level of abstraction required for analysing licensing issues. A couple of alternative theories of the legal concept of a derivative work have been modelled using defeasible inference rules in the Legal Knowledge Interchange Format (LKIF). Finally, these theories are used to construct, evaluate and visualize pro and con arguments about whether or not a particular open source license may be used by an example software project.

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.D2.1.3	**Title**	Report on Prototype Decision Support System for OSS License Compatibility Issues
Work Package Number	01.03.10	**Title**	Compatibility
Activity Number	A1	**Title**	Legal Issues

Date of delivery	**Contractual**		**Actual**	
Status	Version 1.0, dated 9/10/2009		final	
Nature	Report			
Dissemination Level	Public			
Abstract (for dissemination)	Building on Semantic Web technology and our prior theoretical and practical work on the Carneades argumentation system, we have developed a proof-of-concept, prototype system for helping developers to construct, explore and compare legal theories when analysing open source licensing issues in particular cases. The prototype takes into consideration an analysis of requirements. This analysis concludes that the resolution of open source licensing issues is an argumentative process in which alternative theories of copyright law concepts, such as the concept of a derivative work, together with the facts of particular cases, are constructed and critically evaluated. An ontology of open source licences has been developed, using the Web Ontology Language (OWL) and this ontology has been used to model several popular open source licenses, including the Apache 2.0, BSD and MIT academic licenses, as well as the MPL, EPL and GNU GPL reciprocal licenses. Several variants of the GNU GPL are included in the model, including the GNU AGPL and the GNU LGPL. In addition we have developed an ontology for describing software projects, including various relationships between software entities used by the project, at the level of abstraction required for analysing licensing issues. A couple of alternative theories of the legal concept of a derivative work have been modelled using defeasible inference rules in the Legal Knowledge Interchange Format (LKIF). Finally, these theories are used to construct, evaluate and visualize pro and con arguments about whether or not a particular open source license may be used by an example software project.			
Keywords				

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

Table of Contents

1 INTRODUCTION

The previous Qualipso report of WP1.3, "Report on Problem Scope and Definition about OSS License Compatibility" (Deliverable A1.D2.1.3), surveyed 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 and developed a concrete scenario to illustrate legal issues which need to be considered by the developers of open source software. The basic concepts of copyright law were explained, insofar as they are relevant for license compatibility issues and the kinds of legal sources were surveyed which need to be taken into consideration and interpreted when analysing license compatibility issues. Finally, the report included a brief overview of legal reasoning and argumentation, showing how the resolution of open source license compatibility issues, like all legal issues, is a creative, theory-construction process which is not well-defined and thus cannot be fully automated.

In the current report, selected methods from the state-of-the-art of computational models of law, legal reasoning and argumentation are applied to develop a prototype system for helping developers to construct, explore and compare legal theories when analysing open source licensing issues in particular cases.

The remainder of this report is organized as follows. So that this report is self-contained, Section 2 summarizes the main conclusions about the nature of open source license compatibility problems, from the previous WP1.3 report. Section 3 presents the design of the prototype system for helping developers to analyse open source compatibility issues and explaining how methods from computational models of law, legal reasoning and argumentation are applied. The prototype is illustrated in Section 4, using data based on a real system currently under construction. Section 5 presents conclusions and summarizes the main results.

2 Problem Statement: License Issues and Argumentation

In our previous Qualipso report we illustrated some license compatibility issues which developers must face when combining components subject to different licenses and surveyed the kinds of legal sources, such as statutes, case law and legal principles, which must be taken into consideration when analyzing these issues. We came to the conclusion that open source license compatibility issues cannot be analysed in the abstract, but must be analysed in the light of the particular material facts of a case and the legal norms of the applicable jurisdiction.

In legal 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.

The mainstream view within jurisprudence and legal informatics is that legal reasoning involves the construction, evaluation and comparison of alternatives theories of the law and facts of the case [7,11,23,27]. Typically this takes place in critical dialogues, during which arguments pro and con the alternative theories are put forward by the parties.

Figure 1 illustrates relations between different kinds of legal and factual issues, all of which are resolved by argumentation. In this simple example, 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 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.

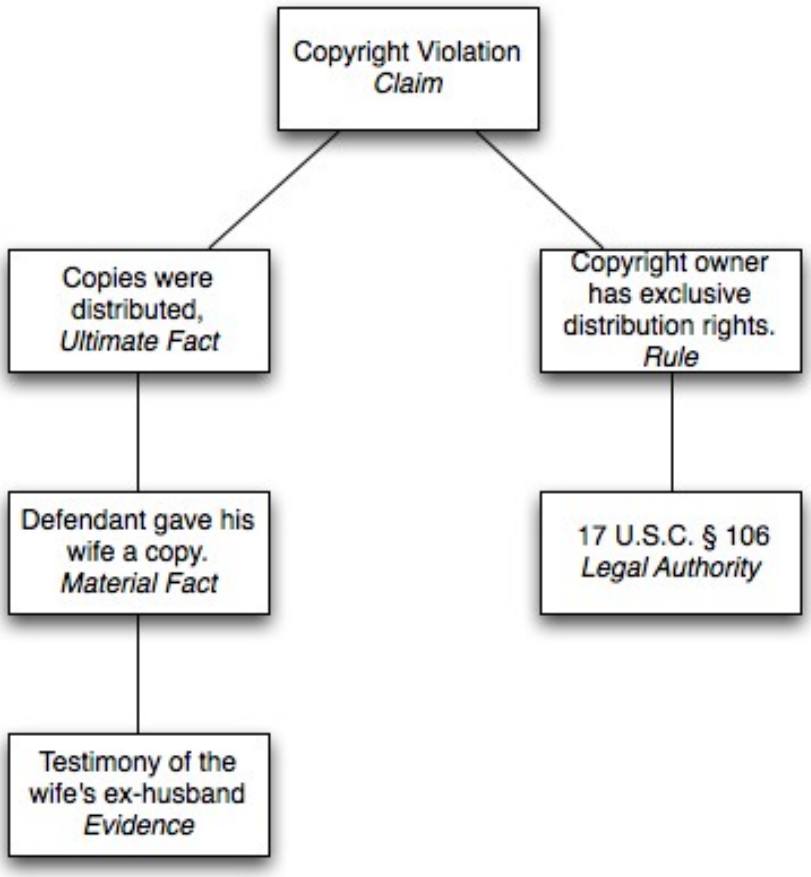

Figure 1: Kinds of Legal and Factual Issues

Arguments are typically enthymemes. That is, 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 by undercutting the argument with an argument claiming that 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.

These general features of legal reasoning lead us to the following use cases for an interactive software tool for helping to analyse open source license compatibility issues:

- Declaring a formal logical language (vocabulary) for software systems, licenses and copyright concepts, including relationships between software works, such as whether one work has been derived from another.
- Representing both strict and defeasible rules of copyright law and parts of the software engineering domain relevant for analyzing open source license compatibility issues. It must be possible to represent and reason with, when analyzing a single case, rules for alternative interpretations of legal sources, such as legislation and case law, rules from various levels within a single legal system (e.g. state and federal law), as well as rules from multiple legal systems (e.g. US and German copyright law). It must also be possible to reason about priority relationships between rules, using principals such as lex superior, and to handle general rules with exceptions.
- Formally defining the terms and conditions of common open source template licenses, including reciprocal licenses, such as the GNU GPL (hereafter simply "GPL") and non-reciprocal licenses, such as the BSD license.
- Describing the material facts of use and derivation relationships between software works, using the formal language, as well the license or licenses which apply to each work.
- Representing evidential arguments about the material facts, such as arguments from witness testimony.
- Constructing legal arguments from the strict and defeasible rules about the legal issues of a case, such as whether dynamic linking results in a derivative work.
- Evaluating a set of arguments, taking into consideration applicable proof standards, the allocation of the burden of proof and assumptions about the beliefs of the "audience", i.e. the persons responsible for making the decision, such as a judge or jury, in order to estimate whether or not some legal or factual claim should be acceptable to the audience.
- Using argumentation schemes to help reveal hidden premises of arguments and ask sensible critical questions.
- Visualizing relationships among a set of arguments, to obtain a comprehensible overview and summary of the issues.

- Determining minimal and consistent set of propositions which, if accepted as true by the relevant audience, would be sufficient to prove or disprove a given claim, depending on one's goal.

Again, as should be clear from these use cases, the system is not conceived to be an algorithm or automatic theorem prover for computing or deriving “the right answer” to open source license compatibility issues, but rather as an interactive tool for helping users to construct, evaluate and visualize competing legal and factual arguments.

3 System Design: Overview of the Carneades Argumentation System

The prototype application we have developed for analyzing open source license compatibility issues has been built using our Carneades software system.[1] Carneades is an interactive application for argument construction, evaluation and visualization, integrating an knowledge-based inference engine and an argument mapping tool. Here we present an overview of the current version of Carneades, explaining how the tools can be used to support argumentation tasks and providing some technical information about how they have been implemented.

Carneades is a set of open source software tools for supporting a range of argumentation tasks, based on a mathematical model of Doug Walton's philosophy of argumentation and developed in collaboration with him over the course of several years, beginning in 2006 [14,15]. Work on Carneades is a research vehicle for studying argumentation from a more formal, computational perspective than is typical in the field of informal logic, and for developing prototypes of tools designed to be useful for supporting real-world argumentation in practice.

We began this project by doing a use-case analysis of common argumentation tasks, as illustrated in Figure 2. The logical layer, at the bottom of the diagram, covers the construction of arguments from data, information, models and knowledge. We intend the sources of arguments to be very broad, ranging from sensory data, witness testimony and others kinds of evidence, across arguments from the interpretation of natural language texts, up to purely formal derivations of arguments from propositions expressed in some formal language, such as predicate calculus. We view argumentation schemes [35] not only as a useful tool for reconstructing and evaluating past arguments in natural language texts, but also as templates helping to guide users as they construct, 'invent' or generate their own arguments to put forward in ongoing dialogues [13].

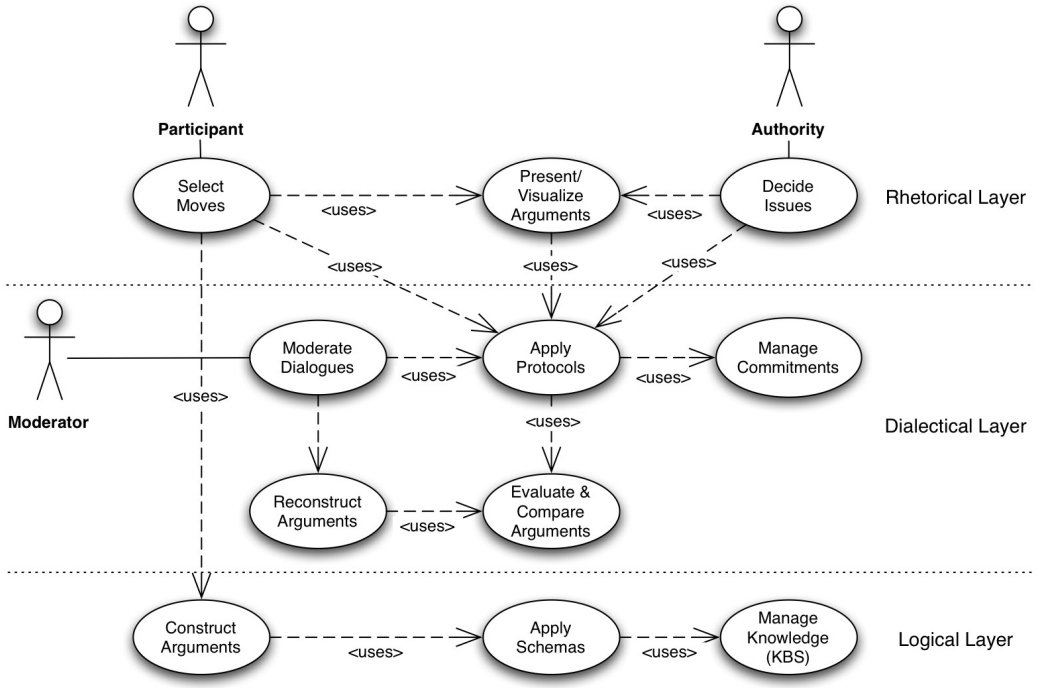

Figure 2: Argumentation Use Cases

[1] http://carneades.berlios.de

The dialectical layer, in the middle of the diagram, covers tasks relevant for comparing and aggregating potentially conflicting or competing arguments, put forward by opposing parties in argumentation dialogues, such as legal procedures before courts. Procedural rules, often called 'protocols', regulate the allocation the burden of proof among the parties, the assignment of proof standards to issues, resource limits, such as due dates for replying or limiting the number of turns which may be taken, and criteria for terminating the process, among other matters.

Finally, the rhetorical level, at the top of the diagram, consists of tasks for participating effectively in argumentation dialogues, taking into consideration the knowledge, experience, temperament, values, preferences and other characteristics of audiences, in particular one's opponent in a dispute. However, rhetoric is not only concerned with methods for taking advantage of an opponent to win a dispute. It is also about expressing arguments in clear ways which promote understanding, given the needs of the audience. We include at this level techniques for visualizing sets of interrelated arguments as argument graphs or maps, as a particular class of methods for presenting arguments in ways which promote understanding.

Notice that the application scenarios which interest us, and which we want to support with software tools, are centered around dialogues, typically with two or more parties, in which claims are made and competing arguments are put forward to support or attack these claims. Following Walton, we recognize that there are many kinds of dialogues, with different purposes and subject to different protocols [36].

We begin in the middle, dialectical layer of Figure 2, because it is central to our work, and not just in the diagram. Since the main task of the bottom, logical, layer, is to construct arguments, and the main task of the top, rhetorical, layer is to present arguments, we first need to define what we mean by arguments and how they are evaluated.

Informally, an argument links a set of statements, the premises, to another statement, the conclusion. The premises may be labelled with additional information, about their role in the argument. Aristotle's theory of syllogism, for example, distinguished major premises from minor premises. The basic idea is that the premises provide some kind of support for the conclusion. If the premises are accepted, then the argument, if it is a good one, lends some weight to the conclusion. Unlike instances of valid inference rules of classical logic, the conclusion of an argument need not be necessarily true if the premises are true. Moreover, some of the premises of an argument may be implicit. An argument with implicit premises is called an enthymeme [37].

We developed the mathematical model of argument which serves as the foundation for the Carneades software tools at the dialectical level in a series of papers [12,14,15,17]. Let us focus here on the later, more mature papers. In [15] we presented a formal, mathematical model of argument structure and evaluation which applied proof standards to determine the acceptability of statements on an issue-by-issue basis. The model uses different types of premises (ordinary premises, assumptions and exceptions) and information about the dialectical status of statements (stated, questioned, accepted or rejected) to allow the burden of proof to be allocated to the proponent or the respondent, as appropriate, for each premise separately. Our approach allows the burden of proof for a premise to be assigned to a different party than the one who has the burden of proving the conclusion of the argument, and also to change the burden of proof or applicable proof standard as the dialogue progresses from stage to stage. Useful for modeling legal dialogues, the burden of production and burden of persuasion can be handled separately, with a different responsible party and applicable proof standard for each. Finally,

following [33], we showed another way to formally model critical questions of argumentation schemes as additional premises, using premise types to capture the varying effect on the burden of proof of different kinds of questions.

In [14], we developed this model further, with the aim of integrating the features of prior computational models of proof burdens and standards, in particular the model of [26] into Carneades. The notions of proof standards and burden of proof are relevant only when argumentation is viewed as a dialogical process for making justified decisions. During such dialogues, a theory of the domain and proofs showing how propositions are supported by the theory are collaboratively constructed. The concept of proof in this context is weaker than it is in mathematics. A proof is a structure which enables an audience to decide whether a proposition satisfies some proof standard, where a proof standard is a method for aggregating or accruing arguments. There are a range of proof standards, from scintilla of evidence to beyond reasonable doubt in the law, ordered by their strictness. The applicable standards depend on the issue and the type of dialogue, taking into consideration the risks of making an error. Whereas finding or constructing a proof can be a hard problem, checking the proof should be an easy (tractable) problem, since putting the proof into a comprehensible form is part of the burden and not the responsibility of the audience. Argumentation dialogues progress through three phases and different proof burdens apply at each phase: The burdens of claiming and questioning apply in the opening phase; the burden of production and the tactical burden of proof apply in the argumentation phase; and the burden of persuasion applies in the closing phase.

The Carneades software, which is implemented in a functional style, enables arguments and argument graphs to be represented and proof standards to be assigned to statements in a graph. Argument graphs are immutable and all operations on argument graphs are non-destructive, as dictated by the functional programming paradigm. Every modification to an argument graph, such as asserting or deleting an argument, or changing the proof standard assigned to a statement, returns a new argument graph, leaving the original unchanged. The acceptability of statements in a graph is computed and, if necessary, updated at the time the graph is modified. Dependency management techniques, known from reason maintenance systems [8,9], are used to minimize the amount of computation needed to update the labels of statements in the graph, as changes are made. Querying an argument graph, to determine the acceptability of some statement in the graph, just performs a lookup of the pre-computed label of the statement, and can be performed in constant time. An XML syntax for encoding and interchanging Carneades arguments, inspired by Araucaria's Argument Markup Language [28], has been developed, as part of the Legal Knowledge Interchange Format [10]. The Carneades software is able to import and export argument graphs in the LKIF format.

Argumentation schemes are useful for reconstructing, classifying and evaluating arguments, after they have been put forward in dialogues, to check whether a scheme has been applied correctly, identify missing premises and ask appropriate critical questions. Argumentation schemes are also useful for constructing new arguments to put forward, by using them as templates, forms or, more generally, procedures for generating arguments which instantiate the pattern of the scheme. We elaborated the role of argumentation schemes for generating arguments in a series of papers [13,18,20], focusing on computational models of argumentation schemes studied in the field of Artificial Intelligence and Law for legal reasoning, including Argument from Defeasible Rules, Argument from Ontologies, and Argument from Cases.

The term “rule” has different meanings in different fields, such as law and computer science. The common sense, dictionary meaning of rule [1] is “One of a set of explicit or understood regulations or principles governing conduct within a particular sphere of activity.” It is this kind of rule that we are interested in modeling for the purpose of constructing arguments. In the field of artificial intelligence and law, there is now much agreement about the structure and properties of rules of this type [16,22,25,32]:

1. Rules have properties, such as their date of enactment, jurisdiction and authority.
2. When the antecedent of a rule is satisfied by the facts of a case, the conclusion of the rule is only presumably true, not necessarily true.
3. Rules are subject to exceptions.
4. Rules can conflict.
5. Some rule conflicts can be resolved using rules about rule priorities, e.g. lex superior, which gives priority to the rule from the higher authority.
6. Exclusionary rules provide one way to undercut other rules.
7. Rules can be invalid or become invalid. Deleting invalid rules is not an option when it is necessary to reason retroactively with rules which were valid at various times over a course of events.
8. Rules do not counterpose. If some conclusion of a rule is not true, the rule does not sanction any inferences about the truth of its premises.

One consequence of these properties is that rules cannot be modeled adequately as material implications in predicate logic. Rules need to be reified as terms, not formulas, so as to allow their properties, e.g. date of enactment, to be expressed and reasoned about for determining their validity and priority.

In the Carneades software, methods from logic programming have been adapted and extended to model legal rules and build an inference engine which can construct arguments from rules. Rules in logic programming are Horn clauses, i.e. formulas of first-order logic in disjunctive normal form, consisting of exactly one positive literal and zero or more negative literals. The positive literal is called the ‘head’ of the rule. The negative literals make up the ‘body’ of the rule. A rule with an empty body is called a ‘fact’. In logic programming these rules are interpreted as material conditionals in first-order logic and a single inference rule, resolution, is used to derive inferences. Since there is no way to represent negative facts using Horn clauses, rules do not counterpose in logic programming, even though they are interpreted as material conditionals and the resolution inference rule is strong enough to simulate modus tollens. In Carneades, we do not interpret rules as material conditionals, but as domain-dependent inference rules. Both the head and body of rules are more general than they are in Horn clauses. The head of a Carneades rule consists of a set of literals, i.e. both positive and negative literals. The body of a Carneades rule consists an arbitrary first-order logic formula, except that quantifiers and biconditionals are not supported. Variables in the body and head of a rule are interpreted as schema variables. Using de Morgan’s laws, Carneades compiles rules into clauses in disjunctive normal form. Given an atomic proposition P, a rule can be used to construct an argument pro or con P if P or ¬P, respectively, can be unified with a literal in the head of the rule.

The burden of proof for an atomic proposition in the body of a rule can be allocated to the opponent of the argument constructed using the rule, by declaring the proposition to be an exception. The syntax of rules has been extended to allow such declarations. Similarly, a proposition in the body of a rule can be made assumable, without proof,

until it has been questioned by the opponent of the argument. These features make it possible to use Carnedes rules to model a broad range of argumentation schemes, where exceptions and assumptions are used to model the critical questions of the scheme. Whether a critical question should be modeled as an exception or an assumption depends on whether the "shifting burden" or the "backup evidence" theory of critical questions is more appropriate [15] .

In computer science, an ontology is a representation of concepts and relations among concepts, typically expressed in some decidable subset of first-order logic, such as description logic [2]. Such ontologies play an important role in integrating systems, by providing a formal mechanism for sharing terminology, and also in the context of the so-called Semantic Web [6] for providing machine-processable meta-data about web resources and services. There is a World Wide Web standard for modeling ontologies in XML, called the Web Ontology Language (OWL) [24]. The Carneades software includes a compiler from OWL ontologies into Carneades rules, based on the Description Logic Programming (DLP) mapping of description logic axioms into Horn clause rules [21].

Ontologies and rules may be used together to construct arguments with Carneades. LKIF uses OWL to define the language of individual, predicate and function symbols, represented as Uniform Resource Identifiers (URIs), which may be used in rules. URIs provide a world-wide way to manage symbols, avoiding ambiguity and name clashes. This enables very large knowledge bases to be constructed, in a distributed and modular way. LKIF files can import OWL ontologies and, recursively, other LKIF files.

We have been experimenting with methods for visualizing Carneades argument graphs and designing graphical user interfaces for working with argument graphs. An important difference between our work and most prior work on argument visualization, with the exception of [34], is that our diagrams are views onto a mathematical model of argument graphs and the user interfaces provide ways to modify, control and view the underlying model. Argument diagramming software for Wigmore [38], Beardsley/Freeman [5] and Toulmin [31] diagrams, such as Araucaria [28], lack this mathematical foundation. Essentially, the diagrams are the models in these other systems, rather than views onto a model.

Our approach gives us much freedom to experiment with different diagramming methods and user interfaces for manipulating Carneades argument graphs, without changing the underlying model of argument. In [19], we described a couple of different approaches, including one which is very close to Wigmore's style of argument diagramming.

4 Prototype: A Tool for Analyzing Open Source Licensing Issues

In this section we present a prototype of a software tool, built using Carneades, for helping developers to analyse open source license compatibility issues. We start by developing a simple ontology of concepts and relations for describing software licenses and use and derivation relationships between works of software. We include in this ontology formal models of some popular open source licenses, such as the GPL and BSD licenses. Next we show how to use the ontology to model relationships between works of software in a software project. We then show how to model some rules of copyright law using the Legal Knowledge Interchange Format, focusing on the issue of what kinds of uses of a work of software produce derivative works. These models are then used to construct, evaluate and visualize arguments about whether or not the project may publish its software using a particular open source license, that is whether a preferred license is compatible with the licenses of the software used by the project. When constructing arguments we illustrate how abduction can be used to focus the search for issues or goals to work on which are helpful for proving, or disproving, depending on one's standpoint and interests, that the license is compatible.

4.1 Ontology for Open Source Licenses

Ontologies in computer science are an advanced kind of data model. Carneades uses the Web Ontology Language (OWL), a World Wide Web standard [24], for representing and interchanging ontologies. OWL which is well supported by various tools, including the open source Protege editor [3].[2] Figure 3 shows a screen shot of the Protege editor being used to view the classes of the ontology we have developed for open source licenses.

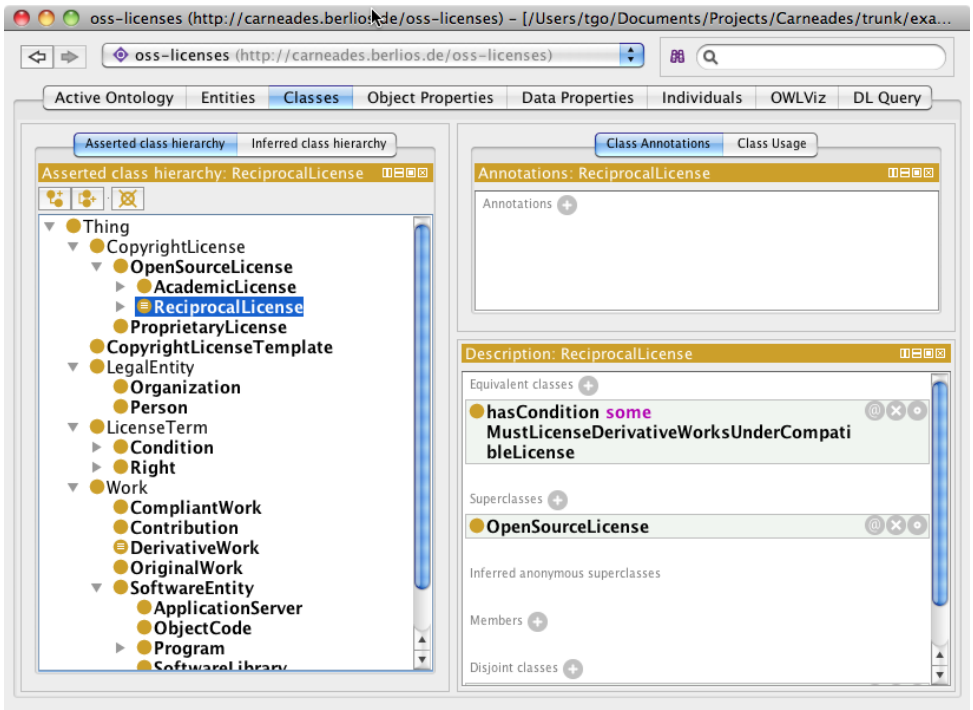

Figure 3: Protege Ontology Editor

The top-level classes, i.e. subclasses of the root `Thing` class, are:

[2] http://protege.stanford.edu/

- `CopyrightLicense`. Individual licenses, with which a particular legal entity, the licensor, grants rights to another legal entity, the licensee.
- `CopyrightLicenseTemplate`. Open source license templates, such as the GPL or BSD. A particular copyright license can be an instance of such a template.
- `LegalEntity`. Legal persons, such as humans, corporations and associations.
- `LicenseTerm`. The rights granted by a license and the conditions of the license, limiting the rights granted.
- `Work`. Various kinds of intellectual products protected by copyright, including software.

Two subclasses of `CopyrightLicense` have been defined:

- `OpenSourceLicense`. This class in turn has `AcademicLicense` and `ReciprocalLicense` subclasses.
- `ProprietaryLicense`

The `CopyRightLicense` class has the following properties:

- `grantsRight: CopyrightLicense × Right`
- `hasCondition: CopyrightLicense × Condition`
- `hasLicensee: CopyrightLicense × LegalEntity`
- `hasLicensor: CopyrightLicense × LegalEntity`
- `instanceOfTemplate:`
  `CopyrightLicense × CopyrightLicenseTemplate`

The `CopyrightLicenseTemplate` class has a property for license compatibility:

- `isCompatibleWith:`
  `CopyrightLicenseTemplate × CopyrightLicenseTemplate`

Notice we have defined this property for license templates, not for individual licenses. Whether or not two particular licenses are compatible with each other can be derived from the compatibility of their templates.

The `isCompatibleWith` property is reflexive (every license template is compatible with itself), but not symmetric. A template license may be compatible with another template license without the reverse necessarily being the case. For example, software which is derived from LGPL software may be licensed using the GPL, but not vice versa.

Open source copyright license templates, such as the GPL, are modelled in two parts in this ontology:

1. By *individuals* of the `CopyrightLicenseTemplate` class:
   a) `ApacheLicense2.0Template`
   b) `BSD_Template`
   c) `EPL_Template`
   d) `GPL_Template`

e) LGPL_Template

f) MIT_License_Template

g) MPL_Template

2. By *subclasses* of the OpenSourceLicense class:

a) AcademicLicense

- ApacheLicense2.0
- BSD
- MIT_License

b) ReciprocalLicense

- EPL
- GPL
- LGPL
- AGPL
- MPL

These two parts of the model of each template license are linked together with an axiom stating that a license which is an instance of a given template is also an instance of the appropriate class, and vice versa. For example, the BSD class is linked to the BSD_Template instance with the following equivalence axiom:

```
BSD ≡ instanceOfTemplate value BSD_Template
```

The reason for modelling each template license as both an instance and a class is that, in order to analyse license compatibility issues, we need some way to reason about whether a license template is compatible with the licenses of the software entities used by a project. For this the license templates need to be individuals, not just classes, because the underlying logic of the Web Ontology Language is description logic, which is semantically a subset of first-order logic. A second-order logic would be needed for reasoning about classes directly.

The ontology also provides classes modelling the rights granted and conditions of licenses and license templates. The following rights and conditions, from the Apache License 2.0, have been included in the ontology thus far:

1. LicenseTerm

a) Right

- MayAcceptWarrantyOrAdditionalLiability
- MayAddYourOwnCopyrightStatementToYourModifications
- MayCopy
- MayDistributeDerivativeWorksInObjectForm
- MayDistributeDerivativeWorksInSourceForm
- MayDistributeOriginalWorkInObjectForm
- MayDistributeOriginalWorkInSourceForm
- MayProduceDerivativeWorks
- MayProvideAdditionalOrDifferentLicenseTermsToYourModifications

- MayPubliclyDisplay
- MayPubliclyPerform
- MaySublicense

b) Condition

- LimitedLiability
- MustDistributeCopyOfLicense
- MustDistributeCopyOfNoticeText
- MustLicenseDerivativeWorksUnderCompatibleLicense
- MustMarkModifications
- MustOfferSourceCode
- MustRetainNoticesInSourceDistributions
- NoPermissionToUseTrademarks
- ProvidedWithoutWarranties

Superclass axioms are used to express that instances of a particular class of license grant certain rights and are subject to certain conditions. Let us illustrate this using the Apache License 2.0. The ApacheLicense2.0 class is defined to be a subclass of the following Right and Condition classes:

- grantsRight some MayAcceptWarrantyOrAdditionalLiability
- grantsRight some MayAddYourOwnCopyrightStatementToYourModifications
- grantsRight some MayCopy
- grantsRight some MayDistributeDerivativeWorksInObjectForm
- grantsRight some MayDistributeDerivativeWorksInSourceForm
- grantsRight some MayDistributeOriginalWorkInObjectForm
- grantsRight some MayProduceDerivativeWorks
- grantsRight some MayPubliclyDisplay
- grantsRight some MayPubliclyPerform
- grantsRight some MaySublicense
- hasCondition some LimitedLiability
- hasCondition some MustDistributeCopyOfLicense
- hasCondition some MustDistributeCopyOfNoticeText
- hasCondition some MustMarkModifications
- hasCondition some MustOfferSourceCode
- hasCondition some MustRetainNoticesInSourceDistributions
- hasCondition some NoPermissionToUseTrademarks
- hasCondition some ProvidedWithoutWarranties

Any license which is an instance of the Apache License 2.0 template entails these rights and conditions. More formally, if an individual is a member or the class

instanceOfTemplate value ApacheLicense2.0Template

then it is, because of the equivalence axiom

ApacheLicense2.0 ≡ instanceOfTemplate value ApacheLicense2.0_Template

also a subclass of the `Right` and `Condition` classes of which the `ApacheLicense2.0` is a subclass.

A license which has all of the rights and conditions of the Apache License 2.0, however, is not necessarily an instance of the `ApacheLicense2.0` class. For this to be the case it must also be an instance of the Apache License 2.0 template. That is, it must express these license terms using exactly the same language as the Apache License 2.0 template.

That said, this way of modelling license terms in a uniform way, across templates, does make it possible to use an OWL reasoner to automatically classify license templates by their terms and to find license templates with preferred or selected terms. For example, suppose you are interested in academic licenses which grant you the right to make copies. To find such licenses, you can define a class in Protege, let's call it `PreferredLicense`, which is equivalent to:

```
AcademicLicense and grantsRight some MayCopy
```

Figure 4 shows a screenshot of Protege showing how this is done.

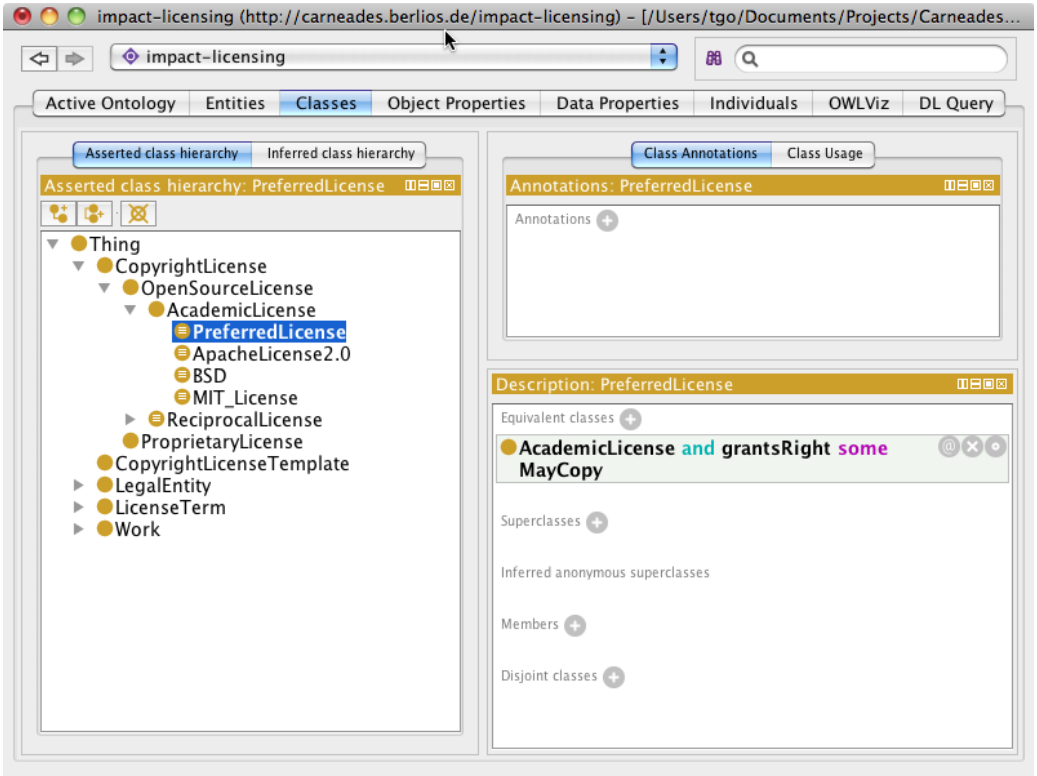

Figure 4: Definition of Preferred License

Protege can then use an OWL reasoner, such as Pellet[3] or Fact++[4] to find licenses which satisfy these conditions. The result is shown in Figure 5. In this example, only the ApacheLicense2.0 class was found, because the terms and conditions of other licenses have yet to be modelled in the ontology.

[3]http://clarkparsia.com/pellet

[4]http://owl.man.ac.uk/factplusplus/

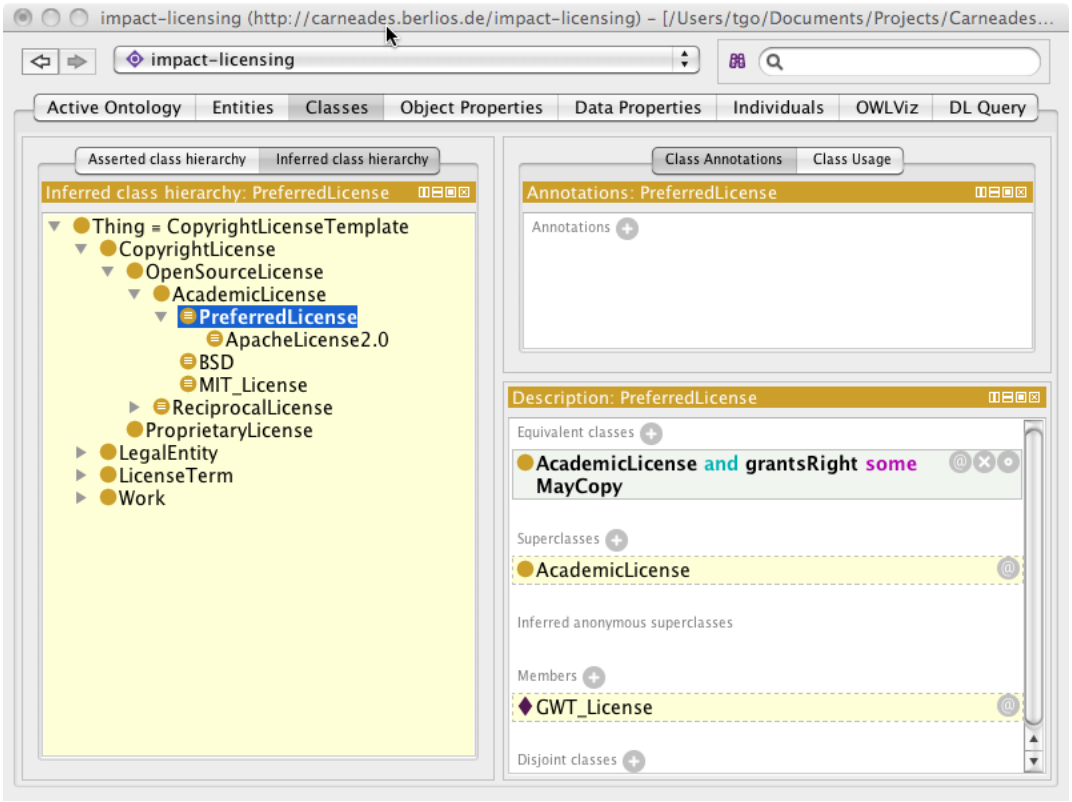

Figure 5: Inferred Preferred Licenses

4.2 Ontology for Software Systems

Next we present the classes and properties of the ontology designed for modelling relationships between software used in a project. The main class, `Work`, models all works protectable by copyright. The `SoftwareEntity` subclass of `Work` is intended to cover all kinds of software artefacts, including not only source and object code, but also more abstract entities such as APIs and specifications. Currently the ontology includes the following subclasses of `SoftwareEntity`, in alphabetical order:

- `ApplicationServer`
- `ObjectCode`
- `OperatingSystem`
- `Program`
  - `RichInternetApplication`
- `SoftwareLibrary`
- `SoftwareService`
- `SourceCode`
- `Specification`
  - `API`
  - `AbstractMachine`
  - `ProgrammingLanguage`

These classes are not intended to be complete, at least not in this prototype, but have been included as needed to model the software entities used in the example project.

Keep in mind that classes in OWL need not be disjunct. Thus, without further axioms in the ontology, a particular software entity can be, for example, be an instance of both the `Program` and `ObjectCode` classes.

The main property of software entities of interest for license compatibility issues is the `isDerivedFrom` property, expressing that one entity has been derived from another. This is a legal issue which will depend on the jurisdiction and the interpretation of the governing law by the courts. The ontology includes properties for representing various ways that software can `use` other software. These properties are not from the domain of copyright law, but rather from the domain of software engineering.

- `uses: Work × Work`
  - `compiledBy: SoftwareEntity × SoftwareEntity`
  - `implementedIn: SoftwareEntity × ProgrammingLanguage`
  - `implements: SoftwareEntity × Specification`
  - `linksTo: SoftwareEntity × SoftwareLibrary`
    - `linksDynamicallyTo: SoftwareEntity × SoftwareEntity`
    - `linksStaticallyTo: SoftwareEntity × SoftwareEntity`
  - `modificationOf: Work × Work`
  - `runsOnOperatingSystem: SoftwareEntity × OperatingSystem`
  - `servedBy: Work × SoftwareEntity`
  - `usesService: SoftwareEntity × SoftwareService`
  - `usesSpecification: SoftwareEntity × Specification`

In legal terms, they provide the means to represent the *material facts* of a case. The legal question is whether a particular use of software, such as linking, is sufficient to create a derivative work. In legal jargon, the question is whether a material fact, linking, can be *subsumed* under a legal concept. Or, more formally, using the properties of the ontology, whether `linksTo` is subsumed by, i.e. a subproperty of, `isDerivedFrom`. None of these use relations has been defined in the ontology as a subproperty of `isDerivedFrom`, because these legal issues have not been resolved, at least not universally, in all jurisdictions, and we want to leave room to argue about these issues. A possible exception might be the `modificationOf` property, which is used to represent software created by textually modifying the source code of existing software. It may be that there is universal agreement, in all jurisdictions, that such modifications create derivate works, with not exceptions. If this is beyond doubt, the `modificationsOf` property could be defined as a subproperty of the `isDerivedFrom` property in the ontology.

An interesting legal issue may be whether the `use` or `isDerivedFrom` properties are transitive. If a software entity X uses Y and Y uses Z, does X also use Z? Similarly, if X is derived from Y and Y from Z, is X also derived from Z? Our intuitions tell us that

the `use` relation, in the domain of software engineering, is transitive but that the `isDerivedFrom` property, which is legal relation, may not be. But since we are not sure, we have not asserted in the ontology that either property is transitive.

In addition to these use relations, the ontology includes properties for representing the license templates which are compatible with the licenses of any works from which it is derived, and for recording the licenses which have been issued for the work:

- `mayUseLicenseTemplate: Work × CopyrightLicenseTemplate`
- `hasLicense: Work × CopyrightLicense`

4.3 Example Model of a Software Project

Figure 6 shows a graph visualizing relationships between software entities of a hypothetical project, roughly based on the Clojure port of Carneades currently being developed. The actual system may be different, since this is work in progress and subject to change.

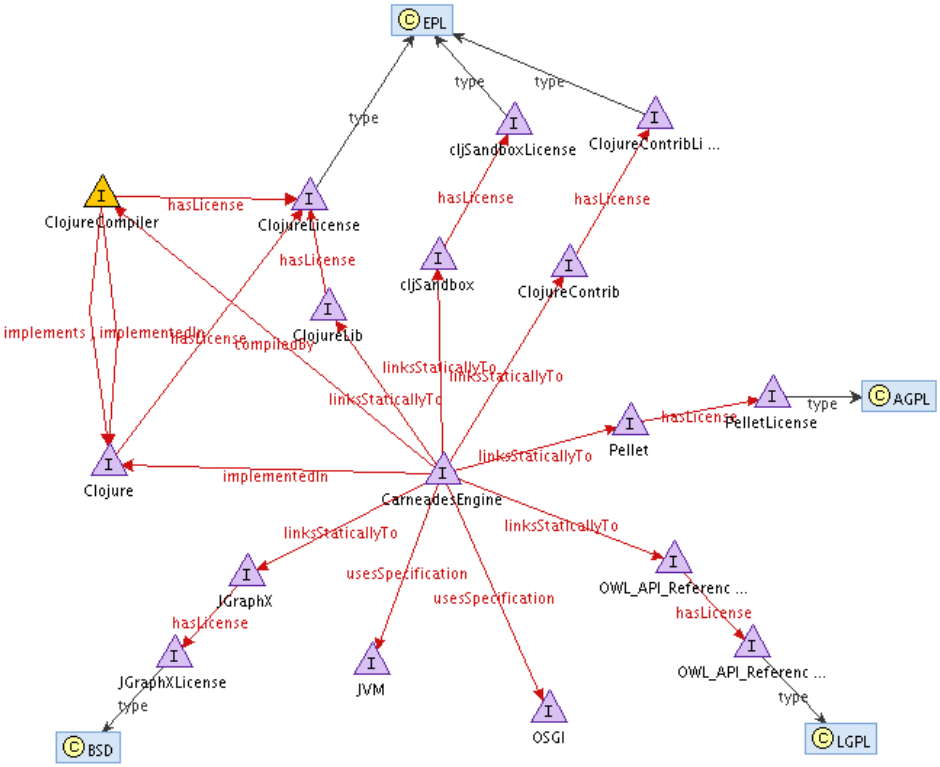

Figure 6: Relationships Between Software Entities of the Example Project

The diagram visualizes individuals in an OWL model of the project, using the ontology presented above, and was creating using a program for visualizing OWL and RDF files called RDF-Gravity.[5]

For our purposes here, it is not necessary to explain all of the software entities and relationships shown in the figure. Let us focus on the Carneades engine, (`CarneadesEngine`), shown in the middle of the figure. The Carneades engine is a software library implemented in the Clojure programming language.[6] It is represented in the model as:

```
CarneadesEngine
      type: SoftwareLibrary
      implementedIn: Clojure
      compiledBy: ClojureCompiler
      linksStaticallyTo: ClojureLib
      linksStaticallyTo: ClojureContrib
      linksStaticallyTo: cljSandbox
      linksStaticallyTo: JgraphX
      linksStaticallyTo: OWL_API
      linksStaticallyTo: Pellet
      usesSpecification: JVM
```

[5]http://semweb.salzburgresearch.at/apps/rdf-gravity/

[6]http://clojure.org

```
      usesSpecification: OSGI
```

Since the example is designed to illustrate the process of using the prototype to help developers select compatible open source licenses, the `CarneadesEngine` does not yet have a license in the model.

The Clojure compiler creates byte code for the Java Virtual Machine (JVM):

```
Clojure
      type: ProgrammingLanguage
      hasLicense: ClojureLicense
ClojureLib
      type: SoftwareLibrary
      hasLicense: ClojureLicense
ClojureCompiler
      type: Program
      hasLicense: ClojureLicense
      implements: Clojure
ClojureLicense
      type: EPL
```

The Clojure compiler is licensed using the EPL. Notice that, in the model, the Clojure individual does not have either the `EPL`, which is a class, or the `EPL_Template` instance as its license directly, but rather has its own, unique license, which we've named `ClojureLicense`. This license is however a member of the `EPL` class, which, as you may recall, is equivalent to the class:

```
instanceOfTemplate value EPL_Template
```

This way of modeling licenses allows each instance of a template license to have its own licensor and licensee, which is necessary since the licensor of software licensed using a template license is the copyright owner of the software and not the owner of the template license, such as the Eclipse Foundation, the owner and maintainer of the EPL template license.

Other libraries used by the Carneades engine include CLJ Sandbox, JgraphX, Pellet, and the reference implementation of the OWL API:

```
cljSandbox
      type: SoftwareLibrary
      hasLicense: cljSandboxLicense
cljSandboxLicense:
      type: EPL
JgraphX
      type: SoftwareLibrary
      hasLicense: JgraphXLicense
JgraphXLicense
      type: BSD
Pellet
      type: SoftwareLibrary
```

```
      hasLicense: PelletLicense
PelletLicense
      type: AGPL
OWL_API_ReferenceImplementation
      type: Software Library
      hasLicense: OWL_API_ReferenceImplementationLicense
OWL_API_ReferenceImplementationLicense
      type: LGPL
```

The EPL and AGPL are reciprocal licenses, so central licensing questions will be whether the Carneades engine must be licensed using either the EPL or AGPL as well, and if one of these licenses is chosen whether the libraries which use the other license still may be used. Legally, this will depend on whether the linking of software to a library causes the software to be a derivative work of the library. We return to this question in the next section, after showing how alternative interpretations of copyright law can be modelled using LKIF rules.

4.4 Rules

The Web Ontology Language, OWL, is based on description logic, which is semantically a decidable subset of first-order logic. This means that the inferences of an OWL reasoner are *strict*: if the axioms of an OWL ontology are true in some domain, then all of the inferences made by a (correctly implemented) OWL reasoner are necessarily also true, without exception. While OWL is very powerful and useful, it is not sufficient for modelling and reasoning about legal norms, such as the rules of copyright law, in a maintainable and transparent way. Legal rules are typically organized as general rules subject to exceptions. Arguments made by applying legal rules are *defeasible.* Their conclusions can be defeated with better counterarguments. Various legal rules may conflict with each other. Theses conflicts are resolved using legal principals about priority relationships between rules, such as the principal of *lex superior*, which gives rules from a higher authority, such as federal law, priority over rules from a lower authority, such as state law. Even when it is possible to reconstruct the meaning of a set of legal rules in OWL, doing so sacrifices maintainability, as rules change, as well as transparency and understandability, because it is difficult to show that the reconstruction is correct and difficult to show how inferences are sanctioned by the authoritative legal sources.

Thus we model legal rules using a defeasible rule language which has been developed especially for this purpose, as part of the Legal Knowledge Interchange Format (LKIF), and use OWL for more limited purposes: 1) to declare the language of unary and binary predicate symbols (classes and properties, in OWL terminology) of the application domain; and 2) to define relationships between these predicates, using OWL axioms, which are judged to be universally true and beyond dispute in the domain. It is a matter of judgement and experience where to draw the line between the parts of the domain which are modelled in the ontology, using OWL, and which are modelled using the LKIF rule language.

Here we illustrate the LKIF rule language by modelling some interpretations of the rules of copyright law, as it pertains to open source licensing issues. Since opinions differ

about how to interpret copyright law in the context of open source licensing issues, for example about whether or not linking to a software library creates a derivative work, an important feature of our approach is the ability to include alternative interpretations in a single model, and to construct and compare competing arguments from these alternative formulations of the rules when analysing licensing issues of a software project.

We begin with the general rule that the copyright owner of software may license the software using any license template he chooses.

```
<rule id="DefaultLicenseRule">
 <head>
   <s pred="&oss;mayUseLicenseTemplate">
     <v>SE</v> may be licensed using the <v>T</v> template
   </s>
 </head>
</rule>
```

Since LKIF is an XML schema, rules are represented in XML. This particular rule has a head (conclusion) but no body (conditions). Even though the rule has no conditions, inferences made using this rule are not necessarily or universally true, but remain defeasible. We will make use of this feature to express exceptions to this general rule below.

The rule has been assigned an identifier, `DefaultLicenseRule`, which may be used to formulate statements about the rule. That is, rules are objects of the domain model, and may be reasoned about just like other objects. This feature is called "reification" in the field of knowledge representation.

The predicate symbol of the statement (proposition) in the head of the rule is specified using the `pred` attribute. Its value can be the name of a class or property in a OWL ontology, as in this example. The `&oss;` entity reference refers the ontology, using its URI. The entity is defined at the top of the LKIF file, as follows:

```
<!DOCTYPE rb [
 <!ENTITY oss "http://carneades.berlios.de/oss-licenses#">
]>
```

Declaring predicate symbols in ontologies makes it possible to divide up the model of a complex domain theory into modules, with a separate LKIF file for each module. OWL provides a way for ontologies to import the classes and properties of other OWL files, recursively. Similarly, LKIF provides a way to import both LKIF and OWL files. OWL makes it easy to manage predicate symbols across the boundaries of modules and to make sure that symbols in different modules refer to the same class or property when this is desired.

The XML syntax for rules in LKIF is rather verbose and not especially readable. Fortunately, it is easy to write programs for converting XML into more readable formats. Moreover, XML structure editors exist, such as Oxygen[7], which use style sheets to enable authors to edit XML documents directly in a more readable form. Using this feature, the above rule can be displayed in the editor as follows:

```
rule DefaultLicenseRule
     SE may be licensed using the T template
```

In this format, variables are underlined. The predicate symbol from the OWL ontology is not shown, but can be viewed and edited in a separate properties panel when the

[7]http://www.oxygenxml.com/

cursor is placed within the text of the statement. We will use this more readable format for displaying LKIF rules in the remainder of this report.

Next let us formulate an exception to the general rule that any license template may be used for reciprocal licenses:

```
rule ReciprocityRule
     not: SE1 may be licensed using the T1 template
  given
     SE1 uses SE2
     SE2 is licensed under L
     L is reciprocal
     SE1 is derived from SE2
     L is an instance of the T2 template license
     unless: T1 is compatible with T2
```

This reciprocity rule states that a software entity, `SE1`, may not be licensed using a template license, `T1`, if the software is derived from another software entity, `SE2`, which is licensed using a reciprocal license, `L`, unless `L` is an instance of a license template, `T2`, which is compatible with `T1`.

Notice that the conclusion of the rule is negated and that the last condition of the rule expresses an exception ("unless …").

The first condition of the rule, `SE1 uses SE2`, serves a heuristic purpose. It provides control information enabling fully instantiated arguments to be constructed when this rule is applied, without having to first search for arguments for the conditions of the rule. Given a model in OWL of the software entities of a project and their various use relationships, an OWL reasoner can be used to derive all use relationships entailed by the model. That is, the reasoner can construct a set of propositions instantiating the form `SE1 uses SE2`, where each proposition in the set is logically entailed by the model. An inference engine for LKIF rules, such as Carneades, can then iterate over these propositions to apply this rule to create fully instantiated arguments for each member of the set.

We cannot use the `isDerivedFrom` property directly for this heuristic purpose, since it is an open legal issue which kind of use relationships result in derivative works and thus this property is not defined not in the ontology. Moreover, different jurisdictions may interpret the concept of derivate works differently and we have aimed to construct the ontology in a way which is independent of the law of specific jurisdictions.

Let us end this brief overview with rules modelling two conflicting views about whether or not linking creates a derivative work.[8] According to the lawyers of the Free Software Foundation, linking does create a derivate work. Lawrence Rosen, a legal expert on open source licensing issues [29] takes the opposing point of view and argues that linking per se is not sufficient to create derivate works.

```
rule FSFTheoryOfLinking
     SE1 is derived from SE2
  given
     SE2 is a software library
     SE1 is linked to SE2
     The FSF theory of linking is valid

rule RosenTheoryOfLinking
     not: SE1 is derived from SE2
```

[8] http://en.wikipedia.org/wiki/GNU_General_Public_License#Linking_and_derived_works

```
given
   SE2 is a software library
   SE1 is linked to SE2
   The Rosen theory of linking is valid
```

The last condition of each of these rules, requires that the interpretation of copyright law represented by the rule is legally valid. The `valid` predicate is "built-in" to Carneades. It need not be imported from an ontology. In these rules, the validity condition is formulated as an ordinary condition, not an assumption or exception, and thus require whoever uses one of these rules to construct an argument to prove that the particular theory is legally valid. Had these conditions been modelled as exceptions, the other side would have had the burden of proving that the theory is not valid in order to undercut arguments constructed with the rules.

4.5 Arguments

Now let's use the theory of open source licensing issues we have constructed with OWL and LKIF rules to analyse a licensing issue of the hypothetical software project, as presented in Section 4.3. Recall that in the example, the Carneades engine is implemented using the Clojure programming language and links to some Clojure libraries, as well the JgraphX, Pellet and OWL API libraries. The Clojure compiler and libraries are licensed using the EPL. The JgraphX library uses the BSD license. The Pellet library uses the AGPL variant of the GPL. And, finally, the OWL API reference implementation uses the LGPL. All of these license templates, except BSD, are reciprocal. See Figure 6 for an diagram showing relationships between the Carneades engine, these libraries and their licenses.

Suppose we want to analyse whether the Carneades engine may be licensed using the Eclipse Public License (EPL). Using the Carneades editor, an interactive argument mapping (diagramming) tool integrated with an assistant which helps users to construct arguments from ontologies and rules, we can create a argument graph about this issue, as shown in Figure 7.

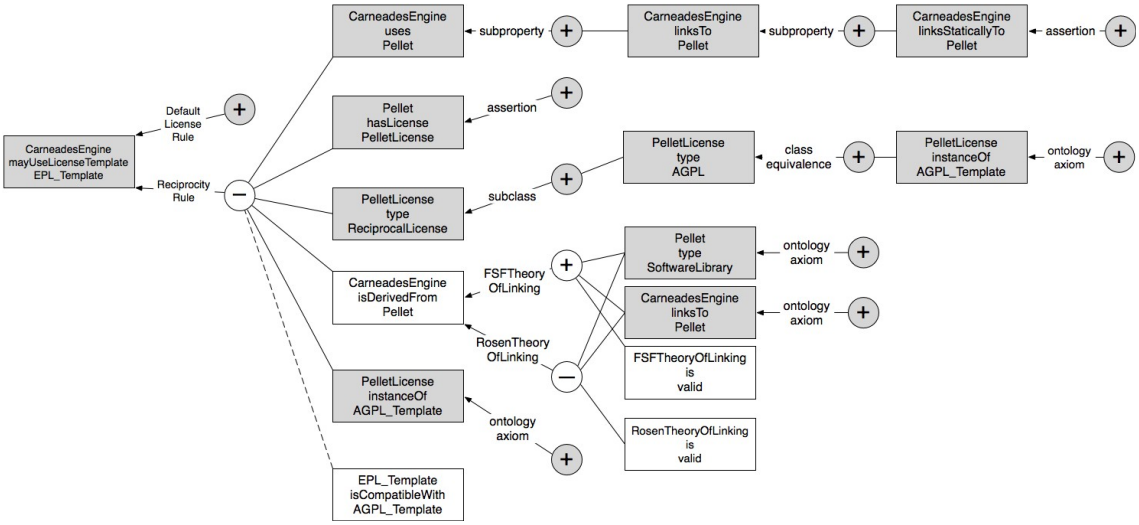

Figure 7: Example Argument Graph

Carneades argument maps are visualizations of argument graphs. An argument graph is a bipartite graph, with statement and argument nodes. Statement nodes are displayed using boxes; argument nodes with circles. The propositional content of a statement node is shown as text inside the box, in either natural language or, as in this example, in some formal language. Here statements are displayed as RDF triples of the form *subject predicate object*. For example, the main issue, about whether the Carneades engine may use the EPL license template, is shown in the box at the far left of the map, with the text:

```
CarneadesEngine mayUseLicenseTemplate EPL_Template
```

The map includes two arguments about this issue, a pro argument and a con argument. Pro arguments are visualized by circles containing a large plus sign; con arguments by circles containing a minus sign. In the example, the pro argument was constructed from the `DefaultLicenseRule`. Recall that this general rule states that, by default, a software entity may be licensed using any template license, with no limitations on the copyright owner. Since this rule has no conditions, the argument which has been constructed using this rule has no premises. Arguments are applicable when all of their premises hold. Arguments with no premises, such as this one, are always applicable. In the figure, applicable arguments and acceptable statements are visualized by filling their circle or box, respectively, with a gray background.

Arguments are linked to their premises and conclusion uses various kinds of edges between the statement and arguments nodes in the diagram. The link from the argument to its conclusion is displayed using an edge with an arrowhead pointing to the conclusion. The links from the argument to its premises do not have arrowheads. Solid lines denote ordinary premises; dotted lines denote exceptions. Negated premises, of which there is no example in the figure, are visualized by adding a crossbar to the premise link. In the figure, we have labelled the link with the arrowhead from the argument to its conclusion with the name of the rule which has been used to construct the argument.

Argument graphs are not restricted to trees. Several arguments may have premises with the same propositional content, i.e. about the same statement. There is an example in Figure 7: The statement

```
PelletLicense instanceOfTemplate AGPL_Template
```

is a premise of two arguments, the con argument constructed using the reciprocity rule, on the left side of the figure, as well as the pro argument about the Pellet license being a AGPL license, to the right. We have duplicated the statement node to simplify the layout of the diagram, but conceptually it is important to keep in mind that these are the same, identical, statements.

Ten con arguments can be constructed using the reciprocity rule in the example project, one for each of the software entities used by the project. This does not mean that all of these arguments are applicable, i.e. that all of the premises of each of these arguments hold. It only means that these are arguments that one might want to consider. Many more arguments could also be constructed using the reciprocity rule, for example by imagining that the software entities used other license templates. Good heuristics are needed to control the search for arguments. Here we are using the factual assertions in the ontology to focus on only those arguments whose material conditions are assumed to be satisfied.

Figure 7, due to legibility and space restrictions, only shows one of these 10 con arguments, the one constructed by applying the reciprocity rule to the Pellet library. This con argument is not (yet) applicable, and thus shown with a white background in the figure, because it is unresolved whether the Carneades engine is derived from the Pellet library. (The last premise of the con argument, about whether the EPL is compatible with the AGPL, is an exception, and thus need not hold for the con argument to go through.)

As can be seen in the figure, the FSF and Rosen theories about whether or not linking creates a derivative work have both been applied, to construct pro and con arguments about whether or not the Carneades engine is derived from Pellet. The facts are undisputed. Pellet is a software library and the Carneades engine, in this hypothetical example, does link to Pellet. The open question is whether either of these two legal theories is valid. If the FSF theory of linking is correct, then the Carneades engine is derived from Pellet and the con argument from reciprocity would be applicable. This con argument would then rebut the argument from the default license rule, with the result that it would no longer be acceptable to use the EPL license.

However further arguments on the other side of this issue could lead us back to the conclusion that the EPL is acceptable. For example, in principal it could be that both the Rosen and the FSF theories of linking are correct, but in different jurisdictions. If the Rosen rule is valid in a higher jurisdiction, then the legal principal of *lex superior* could be used to give it higher priority. In Carneades, this can be done by the giving the argument from the Rosen theory greater weight.

When analysing a case and constructing arguments, it is important to focus your efforts on relevant issues and to choose goals to work on which are promising, given your interests. In a recent conference paper [4], Stefan Ballnat and I presented a model of abduction for Carneades and show how it can be used to support goal selection. Returning to our example of Figure 7, if someone is interested in rebutting the conclusion that the EPL may be used, which is currently acceptable, given the arguments, what issues should he or she focus on next? The model of abduction for Carneades computes minimal sets of statements which, if true (accepted), would make a given statement *in* (acceptable or accepted) or *out* (not in). Since the aim is to rebut the conclusion that the EPL may be used, this conclusion needs to be made out. The minimal sets of statements which, if proved, would achieve this goal are:

- `{FSFTheoryOfLinking is valid}`, and

- {¬CarneadesEngine mayUseLicenseTemplate EPL_Template}

The second of these *positions* simply assumes, without proof, that which needs to proven. (More precisely, it assumes the negation of the statement to be disproven.) Thus this position is uninteresting for our purposes. This leaves the first position, which suggests that, to disprove that the EPL may be used, one should focus on trying to prove the validity of the FSF theory of linking.

To complete the analysis of whether the software developed in the example project may licensed using the EPL, we would need to repeat the procedure illustrated above for each of the other software entities used by the project. If we wanted to evaluate other open source licenses, we would need to do this for each license of interest. Again, the process of using Carneades to evaluate licensing issues is not fully automatic. Carneades is designed as an interactive tool for helping users to construct and evaluate arguments. Here, we have illustrated the main features of this tool by showing how it could be used to help analyse some open source licensing issues.

5 Conclusion

Building on Semantic Web technology and our prior theoretical and practical work on the Carneades argumentation system, we have developed a proof-of-concept, prototype system for helping developers to construct, explore and compare legal theories when analysing open source licensing issues in particular cases. The prototype takes into consideration an analysis of requirements. This analysis concludes that the resolution of open source licensing issues is an argumentative process in which alternative theories of copyright law concepts, such as the concept of a derivative work, together with the facts of particular cases, are constructed and critically evaluated compared. An ontology of open source licences has been developed, using the Web Ontology Language (OWL) and this ontology has been used to model several popular open source licenses, including the Apache 2.0, BSD and MIT academic licenses, as well as the MPL, EPL and GNU GPL reciprocal licenses. Several variants of the GNU GPL are included in the model, including the GNU AGPL and the GNU LGPL. In addition we have developed an ontology for describing software projects, including various relationships between software entities used by the project, at the level of abstraction required for analysing licensing issues. A couple of alternative theories of the legal concept of a derivative work have been modelled using defeasible inference rules in the Legal Knowledge Interchange Format (LKIF). Finally, these theories are used to construct, evaluate and visualize pro and con arguments about whether or not a particular open source license may be used by an example software project.

In the future we hope to find an opportunity to develop the ontology and rulebase further and to validate in pilot applications the suitability of Carneades as a tool for analysing open source license issues. If the validation process is successful, we plan to take steps to publish this application of Carneades as a open source tool for the open source software community.

6 Acknowledgements

I would like to acknowledge the prior work of my former colleague Markus Schmidt on modelling open source licenses using the Web Ontology Language [30], my colleagues working with me at Fraunhofer FOKUS in the Carneades project, in particular Stefan Ballnat and Pierre Allix, the students of my Legal Knowledge-Based Systems project at the University of Potsdam in the winter semester of 2009/2010, who developed their own prototype application for the same problem, as well as Douglas Walton for his inspiration and support and for the opportunity to collaborate with him on computational models of argument over the last few years.

References

1. *New Oxford American Dictionary*. Oxford University Press, 2001.

2. *The Description Logic Handbook — Theory, Implementation and Applications*. Cambridge University Press, 2003.

3. The Protege Ontology Editor and Knowledge Acquisition System. 2007.

4. Ballnat, S. and Gordon, T.F. Goal Selection in Argumentation Processes — A Formal Model of Abduction in Argument Evaluation Structures. *Proceedings of the Third International Conference on Computational Models of Argument (COMMA)*, IOS Press (2010), 51-62.

5. Beardsley, M.C. *Practical Logic*. Prentice Hall, New York, 1950.

6. Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American 284*, 5 (2001), 34-43.

7. Bing, J. Uncertainty, Decisions and Information Systems. In C. Ciampi, *Artificial Intelligence and Legal Information Systems*. North-Holland, 1982.

8. De Kleer, J. A general labeling algorithm for assumption-based truth maintenance. *Proceedings of the 7th national conference on artificial intelligence*, Morgan Kaufmanns Publishers (1988), 188-192.

9. Doyle, J. A Truth Maintenance System. *Artificial Intelligence 12*, (1979), 231-272.

10. ESTRELLA Project. *The Legal Knowledge Interchange Format (LKIF)*. 2008.

11. Engisch, K. *Logische Studien zur Gesetzesanwendung*. C. Winter, 1960.

12. Gordon, T.F. and Walton, D. The Carneades Argumentation Framework – Using Presumptions and Exceptions to Model Critical Questions. *Proceedings of the 6th ECAI Workshop on Computational Models of Natural Argument (CMNA 6)*, (2006).

13. Gordon, T.F. and Walton, D. Legal Reasoning with Argumentation Schemes. *12th International Conference on Artificial Intelligence and Law (ICAIL 2009)*, ACM Press (2009), 137-146.

14. Gordon, T.F. and Walton, D. Proof Burdens and Standards. In I. Rahwan and G. Simari, *Argumentation in Artificial Intelligence*. Springer-Verlag, Berlin, Germany, 2009, 239-260.

15. Gordon, T.F., Prakken, H., and Walton, D. The Carneades Model of Argument and Burden of Proof. *Artificial Intelligence 171*, 10-11 (2007), 875-896.

16. Gordon, T.F. *The Pleadings Game; An Artificial Intelligence Model of Procedural Justice*. Springer, New York, 1995.

17. Gordon, T.F. A Computational Model of Argument for Legal Reasoning Support Systems. *Argumentation in Artificial Intelligence and Law*, Wolf Legal Publishers (2005), 53-64.

18. Gordon, T.F. Constructing Arguments with a Computational Model of an Argumentation Scheme for Legal Rules. *Proceedings of the Eleventh International Conference on Artificial Intelligence and Law*, (2007), 117-121.

19. Gordon, T.F. Visualizing Carneades Argument Graphs. *Law, Probability and Risk 6*, 2007, 109-117.

20. Gordon, T.F. Hybrid Reasoning with Argumentation Schemes. *Proceedings of the 8th Workshop on Computational Models of Natural Argument (CMNA 08)*, (2008), 16-25.

21. Grosof, B.N., Horrocks, I., Volz, R., and Decker, S. Description Logic Programs: Combining Logic Programs with Description Logics. *Proceedings of the Twelth International World Wide Web Conference (WWW 2003)*, ACM (2003), 48-57.

22. Hage, J.C. *Reasoning with Rules – An Essay on Legal Reasoning and its Underlying Logic*. Kluwer Academic Publishers, Dordrecht, 1997.

23. McCarty, L.T. Some Arguments About Legal Arguments. *International Conference on Artificial Intelligence and Law*, (1997), 215-224.

24. McGuinness, D.L. and van Harmelen, F. *{OWL Web Ontology Language} Overview*. 2004.

25. Prakken, H. and Sartor, G. A Dialectical Model of Assessing Conflicting Argument in Legal Reasoning. *Artificial Intelligence and Law 4*, 3-4 (1996), 331-368.

26. Prakken, H. and Sartor, G. A Logical Analysis of Burdens of Proof. In H. Kaptein, H. Prakken and B. Verheij, *Legal Evidence and Proof: Statistics, Stories, Logic*. Ashgate Publishing, 2009, 223-253.

27. Rawls, J. Outline of a Decision Procedure for Ethics. *Philosophical Review*, (1951), 177-197.

28. Reed, C.A. and Rowe, G.W. Araucaria: Software for Argument Analysis, Diagramming and Representation. *International Journal of AI Tools 13*, 4 (2004), 961-980.

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

30. Schmidt, M. Anwendung semantischer Technologien für die Modellierung und Analyse von Lizenzen im Bereich der Open Source Software. 2008.

31. Toulmin, S.E. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.

32. Verheij, B. Rules, Reasons, Arguments. Formal Studies of Argumentation and Defeat. 1996.

33. Verheij, B. Dialectical Argumentation with Argumentation Schemes: An Approach to Legal Logic. *Artificial Intelligence and Law 11*, 2-3 (2003), 167-195.

34. Verheij, B. *Virtual Arguments*. TMC Asser Press, The Hague, 2005.

35. Walton, D., Reed, C., and Macagno, F. *Argumentation Schemes*. Cambridge University Press, 2008.

36. Walton, D. *The New Dialectic: Conversational Contexts of Argument*. University of Toronto Press, Toronto; Buffalo, 1998.

37. Walton, D. *Fundamentals of Critical Argumentation*. Cambridge University Press, Cambridge, UK, 2006.

38. Wigmore, J.H. *A Treatise on the System of Evidence in Trials at Common Law: Including the Statutes and Judicial Decisions of all Jurisdictions of the United States*. Little, Brown and Company, Boston, Massachusetts, USA, 1908.