An Ontology in OWL for Legal Case-based Reasoning

Adam WYNER [1]
Department of Computer Science, University of Liverpool, Liverpool, UK.

Abstract.
In this paper, we present an ontology in OWL for Legal Case-based Reasoning. We outline some of the main motivations for providing an ontology in OWL, then discuss Legal Case-based Reasoning along the lines of CATO and given a method in which the factors of cases are compared and partitioned. An ontology in OWL, developed in the Protege ontology editor, is presented. The ontology is useful as a formal conceptualisation of what one reasons with in Legal Case-based Reasoning. We claim that as legal cases are the input to Legal Case-based Reasoning, the form in which cases are modelled ought to conditioned by the purpose to which they are put.

Keywords.
OWL, Ontology, Legal reasoning, Legal cases, Case-based reasoning

Introduction

In this paper, we present an ontology in OWL for legal case-based reasoning (LCBR), where one reasons from cases that are legal precedents (PCs) to argue for a decision for a plaintiff (P) or defendent (D) in a current case (CC). While ontologies of the law and for legal reasoning are under active development ([6], [9]) and there are references to ontologies for LCBR (cf. [8] and [17]), there is little work which explicitly presents an ontology for LCBR. Moreover, we are not aware of LCBR ontologies which are designed to be web-based and as such support legal decision systems over the internet. Our ontology in the *Ontology Web Language* (OWL) is tailored to LCBR and for deployment over the web.

In the course of modelling LCBR, we necessarily model legal cases. Arguably the *raison d'etre* of modelling legal cases is to provide the instances on which to carry out LCBR (see [14] for a related point). While there may be many aspects of cases which can be modelled, it is questionable that they are relevant unless they are used by some system of LCBR. A similar point can be made with respect to information retrieval of cases [17]. Indeed, we would argue that to model an aspect of cases is to imply that there is such a LCBR system, and it ought to be provided. While we do not consider this issue in its full

[1]Correspond to: Department of Computer Science, University of Liverpool, L69 3BX, UK. Tel.: +44 (0)151 795 4294; Fax: +44 (0)151 795 4235; E-mail: azwyner@liverpool.ac.uk. During the writing of this paper, the author was supported by the Estrella Project (The European project for Standardized Transparent Representations in order to Extend Legal Accessibility (Estrella, IST-2004-027655)).

complexity, our paper is a novel contribution to this discussion, using a relatively simple model of cases in a particular system of LCBR.

As an ontology in OWL for LCBR is novel, we review basic motivations for an ontology in general and an ontology in OWL in particular. We briefly refer to the key works in LCBR as well as present the particular approach to LCBR [16] for which this is the ontology. We then present the ontology, followed by some future work and conclusions. It should be emphasised that this is *an* ontology for LCBR, and there may be other useful ontologies relative to other analyses of LCBR, though some of the components of our proposed ontology may be common to all.

1. Legal Ontologies and Ontology Web Language (OWL)

In this section, we outline some of the main reasons for providing an ontology in OWL and using the ontology development tool *Protege* ([15] and [10]), with particular reference to issues related to ontologies in the legal domain [5].

An ontology is an explicit, formal, and general specification of a conceptualisation of the objects and structural relations between those object in a given domain. It defines a common vocabulary and organization of information which can be shared, tested, and modified by researchers. An ontology makes an assumptions explicit, separates domain knowledge (what are the objects?) from operational knowledge (how do we use the objects?), has a clear structure which supports analysis, is close to the expert's intuitions that underwrite the formal ontology, and identifies structural units which can be reused in other ontologies. In addition, the limitations of the ontology are narrowly proscribed; the model represents a *slice* of a larger domain and limits the reasoning which can be done with respect to it. As a slice of a larger domain, one's ontology can be used as a module along with ontologies of other domains or as a subontology of a more generic ontology.

In providing an ontology, one gives *classes* of objects along with their properties such as the features, attributes, and restrictions that apply to the class. One specifies *subclasses* which inherit properties from the superclass while being further defined in their particular properties. An ontology describes the general classes; given instances of the classes, we have a *knowledge base*. One then applies rules such as *production rules* to elements of the knowledge base.

We have developed our ontology in the Ontology Web Language (OWL) using the *Protege* ontology editor. OWL provides a machine readable ontology which can then be processed by applications; it is designed to underpin development of the *Semantic Web*. OWL provides a range of *flavours*, which are distinct in terms of the richness of the semantic information; each subsort is associated with a degree of logical expressiveness and associated computational properties. However, for our purposes, OWL Lite has been sufficient. The *Protege* ontology editing tool supports systematic development of an ontology. More importantly, *Protege* provides links to reasoners such that one can test one's proposed ontology for *consistency* as well as generate *inferred classes*: *consistency* means that there is a model in which the classes can all be instantiated and there are no instances with inconsistent properties; the *inferred classes* are those which are consistent with the ontology but which one has not explicitly stated and that may be desireable or undesireable classes. We have tested our ontology with the *Pellet-1.3* reasoner, which found the ontology to be consistent. Finally, *Protege* provides plugins to graphical tools which represent the ontology, which facilitates understanding and communication of the model.

As our domain of application is LCBR, we want an ontology which represents those elements of LCBR such as *features*, *cases*, and *case comparisions*. Our ontology is the conceptualisation of the objects used in the argument schemes for LCBR in [16]. The schemes are used to reason with case comparisons in order to provide explanations which justify why a current undecided case ought to be decided in favour of one party or the other based on comparable precedents. As the ontology is web-based and provided in the context of an open-source European Project (ESTRELLA), our object is to allow public use and development of the ontology. In the legal domain, the importance of ontology development is relatively recent, as it has became clear that the application of legal rules is often contingent on the satisfaction of a particular definition and as a range of problems with rule-based systems emerged [5, p.5, 10-11]. Our paper is a novel contribution to this growing body of work.

2. Background on LCBR

There are a variety of approaches to implementations of LCBR, beginning with HYPO [2], and subsequently developing into CATO [1], IBP [7], CABARET [12], and BankXX [13]. More theoretical work on LCBR appears in [11], [3], and [4]. For the purposes of our ontology, we have focussed on the CATO system [1] to which we have applied the case comparison method of [3]. The key object of the ontology is the case comparison. While there are several components that go into the construction of the case comparison, the method itself which generates case comparisons is outside the scope of the ontology, though the requirements of the method inform the construction of the ontology. Thus, to understand the design of the ontology, we outline the elements of the method in this section, focussing primarily on the partitioning of the case factors and the association with the parties of a case. We also provide some examples. The chief reason we have adopted this method is for clarity and familiarity; as ontologies in OWL for other approaches LCBR are developed, presumably they would be *aligned*, *related*, and common elements *abstracted* into an *über* ontology in OWL for LCBR.

CATO [1] represents LCBR using factors, cases, and parties to cases (i.e. P and D). We focus on the factors and how we use them in comparing cases, leaving other aspects for further discussion in the ontology. The factors are those features of a case which are used in making a decision in favour of one of the parties to the case. In addition to a name (which gives an idea of what the factor is about) and an ID (to facilitate reference), the factors are associated with that party of the case which the factor favours, either a D or a P. As in CATO, the factors are organized into a *factor hierarchy* so that one can reason about cases using an *abstract factor*; however, motivation for and use of the factor hierarchy is not relevant to the presentation of the ontology [16].

To illustrate, we use some of the factors, cases, and case comparisons discussed in [16]. Our objective is to give a sense of the method to help understand the ontology rather than to discuss the outcome of the method in every instance. In Table 1, we list factors, the side the factor favours, and the factor parent. The Factor ID gives a handy label to the factor, while the Factor Name gives a brief idea what the factor is about. To say that the factor favours a D or P is to claim that the factor, if present in a case, favours a decision being made for that side. For example, if a case has F1, that is a factor in favour of the case being decided for D. However, whether and how strongly a particular factor favours the decision for a party in a current case depends on the other factors of the case and how the current case factors counterbalance with the factors of a precedent case.

Table 1. Factors

Factor ID	Factor Name	Side	Parent
F1	Disclosure in Negotiations	D	Efforts to Maintain Secrecy
F2	Bribed Employee	P	Questionable Means
F10	Secrets Disclosed to Outsiders	D	Info Known and Available
F12	Outsider Disclosures *Restrict*ed	P	Info Known and Available
F15	Unique Product	P	Valuable Product
F25	Information Reverse Engineered	D	Questionable Means
F26	Used Deception	P	Questionable Means

Table 2 provides hypothetical cases which are variations on *Mason v. Jack Daniels*, in which *Jack Daniels* , a major whiskey manufacturer, is D in a case where *Mason*, a private bar owner, is P; P is sueing D for damages, claiming D stole his secret cocktail recipe and used it in a promotion. Each case contains factors which favour either P (P Factors) or D (D Factors). For example, the case *Vanilla* has the factor *Unique Product*, which favours P since P claims his product was unique, and the factor *Disclosure in Negotiations*, which favours D since D claims the recipe was disclosed in negotiations. The other example cases can be understood in a similar manner.

Turning to the case comparisons, we take as CC a case in which the *outcome is undecided* and compare it to a PC in which the *outcome is decided*. For the moment, we discuss only those PCs decided for P since those which were decided for D can be determined in a similar fashion. Reasoning to a decision in the CC proceeds on the basis of analogy with the PC: as the factors in the PC led to a decision in favour of P and the CC is analogous to the PC, so the CC should also be decided in favour of P. However, the analogy depends on a counterbalancing interplay between the factors, the side each factor favours, and the side favoured in the decision in the PC. To clarify this, we next consider the partition of the factors.

Suppose that we compare the factors of a CC and a previously decided precedent case (PCi). The question is whether on the basis of the comparison we should decide CC for the same party (P or D) as PCi was decided. Given the factors of each case, we form *partitions* of the factors relative to the cases and the side which the factors favour. For instance, in P3, we find those factors in CC which are not in PCi and which favour P. P1

Table 2. Summary of Cases in Example

Case Name	P Factors	D Factors
Vanilla	F15	F1
Bribe	F2, F15	F1
Deceit	F15, F26	F1
Restrict	F12, F15	F1, F10
Reverse	F15	F1, F25

Table 3. Partitions of Factors in CC and PCi

Partition	Biases Decision For	Factors Support
P1	P	P factors in both CC and PCi.
P2	P	D factors in both CC and PCi.
P3	P	P factors in CC not in PCi.
P4	P	D factors in PCi not in CC.
P5	D	D factors in CC not in PCi.
P6	P	P factors in PCi not in CC.
P7	U	Factors not in either CC or PCi.

and P2 represent what is similar in the cases. We assume that a PCi can be a precedent for a CC only so long as these partitions are not empty. In Table 3, we have seven partitions since we compare the intersection of the sets of factors relative to which side is favoured in CC or PCi. In the Table, we represent how the factors support a party and how that partition biases the decision in CC. We discuss each part.

In the ontology, we say that each partition of factors *supports* one side or the other of the case; in Table 3, we see that under the column labelled *Factors Support*, P1 supports P, P2 supports D, and so on. However, we must clearly distinguish between this and *how the partition is used in making a decision in the CC for or against a side*, which we term the *partition bias in CC* and which is presented in the column labelled *Biases Decision For*. *Factors Support* identifies the *side* used for the set, while *Biases Decision For* is based on *the decision in the precedent case*. Recall that we are examining for the moment only PCs which were decided for P; case comparisons which use PCs decided for D follow very similar reasoning, but give rise to different patterns. Suppose a case comparison where we have a non-empty P3, which is the set of P factors in CC that are not in PCi. Here the factors support P. We say that the *partition biases the decision in favour of P* because CC contains more factors for P than PCi, which was decided for P. In contrast, though a non-empty P4 partition has *Factors Support* for D, it biases the decision in favour of P since this partition specifies that the CC has strictly *fewer* D factors; since PC was decided in favour of P with *more* D factors than CC, then CC should be decided in favour of P. In P5, where there are *more* D factors in CC than PCi, the decision is biases in CC towards D. In contrast, P6 biases the decision in CC towards P since while there are more P factors in PCi for P than in CC, those P factors which the cases do share may still be sufficient to decide in favour of P in CC. The partitions in Table 3 represent these biases in the second column. Parallel considerations apply where we use PCs that have been decided in favour of D. However, the presence of the partition only biases the decision and does not decide it since the decision depends on the content of the other partitions and how we reason with them, which is outside the scope of this paper (see [16]).

Using these factors, cases, and case comparsion method, we can provide selected case comparisons as in Table 4. We see in the case comparison *Restrict/Vanilla*, where *Restrict* is the current case and *Vanilla* is the PC, that *Vanilla* can be used as a PC (P1 and P2 are not empty), while P3 are P factors in favour of deciding the current case for P in *Restrict*, but P5 are D factors in favour of deciding the current case for D. The ontology does not consider how these partitions are weighed in coming to a decision on the CC.

Table 4. Selected Case Comparisons

CC/PCi	P1	P2	P3	P4	P5	P6
Bribe/Vanilla	F15	F1	F2	-	-	-
Vanilla/Reverse	F15	F1	-	F25	-	-
Deceit/Bribe	F15	F1	F26	-	-	F2
Restrict/Vanilla	F15	F1	F12	-	F10	-

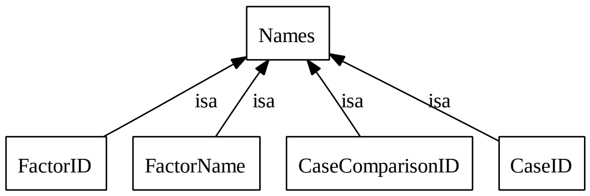
3. LCBR Ontology in OWL

The LCBR ontology has six main classes, which may have their own subclasses. The main classes and their subclasses are mutually disjoint. We discuss each of these classes and their properties and relations. Each class may be comprised of subclasses and have asserted conditions, which relate instances of one class to instances of another class. Subclasses inherit the conditions which hold of every member of the superclass, but may otherwise be distinguished according to other conditions. We start with the less complex classes, then work up to the more complex classes which use the simpler classes. All the classes and subclasses are mutually disjoint.

3.1. Names

We have a class of `Names` which has four subclasses `FactorID`, `FactorName`, `CaseComparisonID`, and `CaseID`. `FactorID` refers to a particular factor, while `FactorName` gives more informative content. As an instance, if we have a `FactorName` such as *Public Disclosure*, meaning that this is a factor in some case, we would give it a `FactorID` such as `F12` in order to abbreviate a reference to it. Every `Case` has a `CaseID`, which is just some way to refer to the particular cases. Similarly, when we come to making *case comparisons* between a CC and a PC, then each comparison has a `CaseComparisonID` to label it. At the bottom level, classes are said to have *instances* of that class. We read Figure 1 from the bottom up as, for example, the class of instances which are `FactorIDs` is a subclass of the class of `Names`. The subclasses are mutually disjoint, meaning there can be no instances which are *both* a `FactorID` and a `CaseID`.

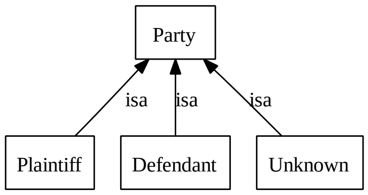
Figure 1. Name Class



3.2. Parties

In Figure 2, we have the class of `Party` with three subclasses `Plaintiff`, `Defendant`, and `Unknown`. The first two are clear. `Unknown` is needed for CCs, which are cases that have not yet been decided in for P or D.

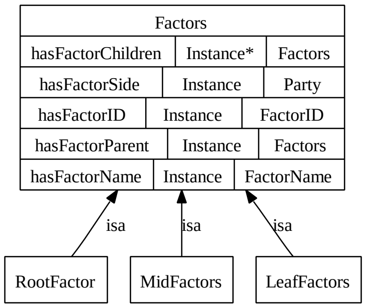
Figure 2. Party Class



3.3. Factors

In Figure 3, we have the `Factor` class, asserted conditions which hold for some factors, and three subclasses. We read from the factor class (the first row) to the property and then an instance of some other class (in some other row). In this figure, we include all the conditions that are found for *some* or all subclasses as we specify below as given in the ontology in OWL. For example, every instance of a `Factor` must have a `FactorID`, a `FactorName` as well as a `FactorSide` which is an instance of some `Party`. The three subclasses `RootFactor`, `MidFactors`, and `LeafFactors` are indicated with the `isa` relation. Subclasses inherit the conditions which hold of the superclass, though additional conditions or restrictions on inherited conditions may apply. We represent the *factor hierarchy* with the `FactorChildren` and `FactorParent` properties. The `RootFactor` has an unknown `FactorSide` and some factors as `FactorChildren`. The `LeafFactors` have some factors as `FactorParent` and no factors as `FactorChildren`. The `MidFactors` have some factors as `FactorParent` and some factors as `FactorChidlren.`

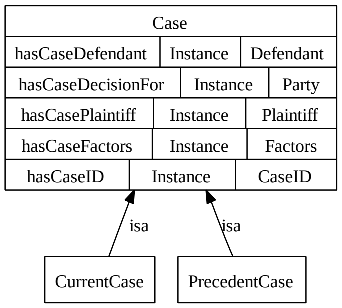
Figure 3. Factor Class


Factors
hasFactorChildren | Instance* | Factors
hasFactorSide | Instance | Party
hasFactorID | Instance | FactorID
hasFactorParent | Instance | Factors
hasFactorName | Instance | FactorName
isa
isa
isa
RootFactor
MidFactors
LeafFactors

3.4. Cases

In Figure 4, we have a class `Case` and two subclasses `PrecedentCase` and `CurrentCase`. Every case has all the properties, so every case has a `CaseDefendant` which is a `Defendant` party, a `CasePlaintiff` which is a `Plaintiff` party, a `CaseDecisionFor` some element of `Party`, a set of `CaseFactors`, and a `CaseID`. The subclass `CurrentCase` has a `CaseDecisionFor` the `Unknown` party, while the subclass `PrecedentCase` has a `CaseDecisionFor` either the `Defendant` or the `Plaintiff` party, but not the `Unknown` party.

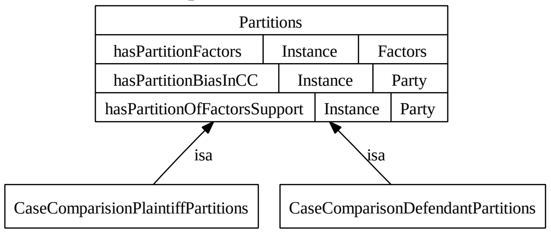
Figure 4. Case Class


Case
hasCaseDefendant | Instance | Defendant
hasCaseDecisionFor | Instance | Party
hasCasePlaintiff | Instance | Plaintiff
hasCaseFactors | Instance | Factors
hasCaseID | Instance | CaseID
isa
isa
CurrentCase
PrecedentCase

3.5. Partitions

In Figure 5, we have the `Partition` class which has two subclasses `Case-ComparisonPlaintiffPartitions` and `CaseComparisonDefendantPart-itions`. These latter two subclasses each have subclasses, which we discuss further below. Every instance of `Partition` has a `PartitionFactors` which is some `Factors`, a `PartitionBiasInCC` which is some `Party`, and a `PartitionOf-FactorsSupport` which is some `Party`. Each of these properties is related to the discussion in Section 2 and further specified when we discuss the subclasses. `Partition-Factors` is just that set of factors created by that particular case comparison of factors. The other two properties give sides as stipulated below.

Figure 5. Partition Class


Partitions
hasPartitionFactors | Instance | Factors
hasPartitionBiasInCC | Instance | Party
hasPartitionOfFactorsSupport | Instance | Party
isa
isa
CaseComparisionPlaintiffPartitions
CaseComparisonDefendantPartitions

Each of `CaseComparisonPlaintiffPartitions` and `CaseComparison-DefendantPartitions` have seven disjoint subclasses, each of which vary from the others in terms of the instantiations of the properties `PartitionBiasInCC` and `PartitionOfFactorsSupport`. We present this as a Table 5.

3.6. Case Comparisons

Our final class appears in Figure 6. It has no subclasses and draws on all the other classes. There is a `CaseComparisonDecisionFor` some `Party` (which presumably can be `Unknown` if no decision is made), a `CaseComparisonID`, a `CurrentCase` and `PrecedentCase`, which are the cases that are used in making the case comparison, and finally `CaseComparisonPartitions`, which are the partitions of factors generated by comparing the current case and PC.

Table 5. Partitions which Bias a Decision and Support a Side

	PartitionBiasInCC	PartitionOfFactorsSupport
Plaintiff Partitions		
1	Plaintiff	Plaintiff
2	Plaintiff	Defendant
3	Plaintiff	Plaintiff
4	Plaintiff	Defendant
5	Defendant	Defendant
6	Plaintiff	Plaintiff
7	Unknown	Unknown
Defendant Partitions		
1	Defendant	Plaintiff
2	Defendant	Defendant
3	Plaintiff	Plaintiff
4	Plaintiff	Defendant
5	Defendant	Defendant
6	Defendant	Plaintiff
7	Unknown	Unknown

Figure 6. Case Comparison Class

CaseComparison		
hasCaseComparisonDecisionFor	Instance	Party
hasCaseComparisonPartitions	Instance	Partitions
hasCurrentCase	Instance	Case
hasPrecedentCase	Instance	Case
hasCaseComparisonID	Instance	CaseComparisonID

4. Future Work

For future work, we intend to examine the issue raised at the onset about the relationship between modelling cases and LCBR. We would continue to use ontology develop to clarify the problems, comparing and contrasting alternative models, as well as making generic ontologies for cases and LCBR. Furthermore, we want to consider how to extend the method of case comparison beyond the legal domain to other issues in case-based reasoning.

5. Conclusion

We have provided a ontology in OWL for LCBR. It contributes to understanding the relationship between modelling LCBR and modelling cases. We have made progress towards several of the objectives outlined in [5]. Domain knowledge (i.e. the ontology) is distinguished from the rules which apply to them found in [16]. Giving an explicit, formal, and general conceptualisation of the LCBR domain, we have a clearer understanding of the elements and properties. The faults of the conceptualisation should be easier to identify; one can easily compare and contrast this ontology with alternatives expressed in the same language. The model is easy to maintain, develop, and reuse. It is also very accessible to the legal expert who is not familiar with formal modelling as familiar structures and relationships are apparent. Furthermore, one can ask questions about the model that are intuitive. We have also narrowly constrained our model to those objects most directly relevant to LCBR. In addition to these longstanding objectives, the ontology is provided in *OWL*, so can be publically available and used as the basis of a markup language. Using *Protege* to develop the language, we can use the reasoning facilities to check that our model is consistent and to generate inferred classes.

References

[1] Vincent Aleven. *Teaching case-based argumentation through a model and examples*. PhD thesis, University of Pittsburgh, 1997.

[2] Kevin Ashley. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals*. Bradford Books/MIT Press, Cambridge, MA, 1990.

[3] Trevor Bench-Capon. Arguing with cases. In A. Oskamp et al., editors, *JURIX 1997*, pages 85–100, Nijmegen, 1997. Gerard Noodt Instituut.

[4] Trevor Bench-Capon and Giovanni Sartor. A model of legal reasoning with cases incorporating theories and values. *Artif. Intell.*, 150(1-2):97–143, 2003.

[5] Trevor J.M. Bench-Capon and Peter R.S. Visser. Deep models, ontologies and legal knowledge based systems. In *Legal Knowledge Based Systems. JURIX 1996: The Nineth Annual Conference.*, pages 3–14. Tilburg University Press, 1996.

[6] Joost Breuker et al. Ontologies for legal information serving and knowledge management. In John Horty, Aspassia Daskalopulu, and Radboud Winkels, editors, *Legal Knowledge and Information Systems: Proceedings of Jurix 2002*, pages 73–82. IOS Press, 2002.

[7] Stefanie Brüninghaus and Kevin D. Ashley. Generating legal arguments and predictions from case texts. In *ICAIL 2005*, pages 65–74, New York, NY, USA, 2005. ACM Press.

[8] John Henderson and Trevor Bench-Capon. Dynamic arguments in a case law domain. In *ICAIL '01: Proceedings of the 8th international conference on Artificial intelligence and law*, pages 60–69, New York, NY, USA, 2001. ACM Press.

[9] Rinke Hoekstra, Joost Breuker, Marcello Di Bello, and Alexander Boer. The lkif core ontology of basic legal concepts. In *Legal Ontologies and Artificial Intelligence Techniques*, Stanford University, Palo Alto, CA, USA, June 2007.

[10] Natalya Noy and Deborah McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Standford University, 2000.

[11] Henry Prakken and Giovanni Sartor. A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law*, 4(3-4):331–368, 1996.

[12] Edwina L. Rissland and David B. Skalak. Cabaret: rule interpretation in a hybrid architecture. *Int. J. Man-Mach. Stud.*, 34(6):839–887, 1991.

[13] Edwina L. Rissland, David B. Skalak, and M. Timur Friedman. Bankxx: Supporting legal arguments through heuristic retrieval. *Artificial Intelligence and Law*, 4(1):1–71, 1996.

[14] Bram Roth and Bart Verheij. Cases and dialectical arguments. an approach to case-based reasoning. In R. Meersman, Z. Tari, and A. Corsaro, editors, *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops. WORM'04: The Second International Workshop on Regulatory Ontologies.*, volume Volume 3292 of *Lecture Notes in Computer Science*, pages 634–651, Heidelberg, 2004. Springer.

[15] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[16] Adam Wyner and Trevor Bench-Capon. Argument schemes for legal case-based reasoning. In to appear, editor, *JURIX 2007*, page to appear, Amsterdam, 2007. IOS Press.

[17] Yiming Zeng, Ruili Wang, John Zeleznikow, and Elizabeth A. Kemp. Knowledge representation for the intelligent legal case retrieval. In Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 3681 of *Lecture Notes in Computer Science*, pages 339–345. Springer, 2005.