Normative Systems: the meeting point between Jurisprudence and Information Technology?

A position paper[1]

Luigi Logrippo
Université du Québec en Outaouais
Gatineau, Québec, Canada
luigi@uqo.ca

Abstract. It is argued that there are many concepts and methods in common between policy systems used in Information Technology and Jurisprudence, i.e. legal theory. These concepts are found in the research area of 'normative systems' which encompasses them and provides a framework for unifying research. It is further argued that advantages can be accrued to both research areas by favoring interchanges of methods and principles in this unifying framework. A distinction is made between norms in rule style and norms in requirements style. Issues of completeness, consistency and conflicts are considered. Concepts that are useful in this research area include defeasible logic and ontologies. Useful tools are theorem provers and model checkers.

1. Background and Motivation

This paper presents the view that legal methodology, Jurisprudence, has many issues and concepts in common with software methodology. There is much that can be learned in both fields by a process of conceptual osmosis, or even convergence. This process will be encouraged by the fact that the behavior of computational agents is increasingly acquiring legal significance. There are several areas in which this is happening: e-business (including e-contracts and security) and IT governance. In these areas, it might even become desirable that laws can be formally translated into computer programs, or that the correspondence of an IT policy with law can be formally audited. For example, a law on privacy may have to be implemented in a set of policies in a language such as XACML [18]. This set of policies may have to be checked for conformance with the law. As the law changes, the XACML policies may have to be changed as well.

A force acting in the converse direction is provided by the fact that computer networks are becoming like social systems, with their own internal norms [20].

At the same time, just as in IT software to help create systems of policies is being developed, in the area of jurisprudence *legislative drafting systems* are being developed

[1] This paper is a slightly improved version of a paper of the same title appeared in H. Fujita, D. Pisanelli (Eds.): New Trends in Software Methodologies, Tools and Techniques. Proceedings of the sixth SoMet_07. IOS Press, 2007, 343-354.

[13]. XML is commonly used for syntactic support of both kinds of systems, however for now formal semantics and semantic validation are not primary goals in either field. Surely, developing the necessary formal models is a long-range research task in both areas.

Motivated by these developments, this paper hinges on the view that information systems policies and legal systems have much in common, in fact are special cases of *normative* systems. We identify a *rule* style and a *requirement* style in both areas. Issues of completeness and consistency are discussed in relation to these two styles. Several concepts and tools of common interest are briefly discussed.

It is important to note that we are not claiming to address all aspects of the systems we are discussing. Legal systems are extremely complex and have aspects that are quite difficult to formalize in any logic or any formal theory, since they have their roots in sociology, history, psychology, ethics and politics [10, 16]. Information systems policies are of many different types for many different applications, but they are all formalized because they are executed by machines. In order to identify similarities, we will schematize and simplify. However we claim and we shall show by examples, that there are common concepts for expressing and analyzing some aspects of these systems.

From a Jurisprudence point of view, we are taking a formalistic approach by which laws are seen as having their own self-contained meaning as pure logical statements, outside of consideration of political, sociological, or moral nature. We recognize of course that these considerations exist and open the way to other types of discussion.

2. Normative systems

In 1993, Jones and Sergot wrote [9]:

> "The general position which we here develop and illustrate is that---at the appropriate level of abstraction---law, computer systems, and many other kinds of organisational structure may be viewed as instances of normative systems. We use the term to refer to any set of interacting agents whose behaviour can usefully be regarded as governed by norms. Norms prescribe how the agents ought to behave, and specify how they are permitted to behave and what their rights are. Agents may be human individuals or collections of human individuals, or computer systems or collections of computer systems. Normative systems include systems of law, abstract models of computer systems, and hybrid systems consisting of human and computer agents in interaction."

We subscribe to this view, with two exceptions. First of all, are normative systems sets of interacting agents (legal institutions), or sets of norms? This question has been extensively debated in philosophy of law and therefore it should be avoided if possible. In this paper, we are mostly interested in sets of norms. Second, this view characterizes norms in terms of the deontic concepts of obligation ('ought to') and permission. This is a very common view, endorsed by the best authorities [10]. However in Section 3 we will see that normative systems can exist without deontic concepts.

There are few attempts to define formally norms and normative systems. Most of these attempts take a limitative view, based on specific formalisms. A much-cited book by Alchourron and Bulygin [1], which claims application to social sciences only, loosely defines norms as statements that relate cases to solutions. As in Jones and Sergot, the solutions are expressed in deontic forms.

We shall take the broad view that normative systems are man-made systems of logical statements, the norms, which relate facts to intended consequences. Their intention is to regulate the functioning of sets of interacting agents. Although they may be expressed in deontic terms, the norms can be translated into fact-consequence form. In this sense, normative systems are similar to rule-based systems. We shall see that the systems in this very general class have some common characteristics that are worth comparing and discussing. The similarities thus recognized can lead to the use of common principles and methods across different types of normative systems.

Examples of policy systems encountered in information technology, to some of which we will make further reference below, are:

- Firewalls and routers
- Telecommunications features, call control
- Information access control systems (e.g. language XACML)
- Security models (Bell-LaPadula, Chinese Wall, RBAC…)
- Web services orchestration and choreography (e.g. language BPEL)
- E-commerce policies and contracts, service-level agreements

3. Norms that are simple rules

As biologists can learn much by studying elementary life forms, we can learn much by studying elementary normative forms.

The Hammurabi code, written about 3,700 years ago, contains norms such as this:

> "If any one steals cattle or sheep, or an ass, or a pig or a goat, if it belong to a god or to the court, the thief shall pay thirty fold; if they belonged to a freed man of the king he shall pay tenfold; if the thief has nothing with which to pay he shall be put to death."

This can be recognized as written in the well-known ECA: <event, condition, action> format which is widely used in data bases, agent systems, etc. [19][2]:

Event = any one steals cattle or sheep, or an ass, or a pig or a goat

Condition = if it belong to a god or to the court

Action = the thief shall pay thirty fold

The Hammurabi code is an early example of coherence in legislative style, since it consists of about 300 articles which are almost all written in ECA format, another witness of the greatness of the Babylonian culture.

On the IT side, let us consider *firewalls*:

```
DROP    all  --  nuisance.com  anywhere
```

This is a rule in a Linux *router* to drop packets having any ("all") protocol that come from node "nuisance.com" and go anywhere. This rule is again in the ECA format, although the condition is empty (conditions compare the incoming events with facts that are known in the context, such as the time of occurrence, or the concepts of 'god' or 'court' in the previous example, which presumably are known in an implicit contextual ontology).

These examples show that, in spite of prevalent opinion to the contrary, normative systems can exist without deontic concepts. In fact, we conjecture that all normative

[2] Such systems of rules could also be read as programs, notably Prolog programs.

systems can be expressed as sets of rules in the ECA format, although this representation may not be finite.

4. Norms in the deontic context

A commonly held view of norms interprets them in a deontic context, for which deontic logic is a frequently used formalization [15]. Deontic logic is a type of modal logic that uses modalities such as *permitted* and *obligatory*, which are mutually related by relationships such as:

> obligatory A = not permitted not A = forbidden not A

In this interpretation, the Hammurabi norm above creates, in the case specified, an obligation to pay thirty fold; the firewall norm creates an obligation to drop all nuisance.com packets.

An early example of legal system that is explicitly based on deontic concepts is Moses' law:

> Thou shalt not steal

In other words, *it is forbidden to steal*. In this normative style, we gain abstraction, since the brief statement just given covers a dozen articles of the Hammurabi code, but we lose specificity: what happens if one steals? How is this norm enforced?

In software engineering terminology, one could think of a *compilation* of the Moses Code into Hammurabi terms, or of a *reverse engineering* of the Hammurabi code into Moses terms.

In IT one encounters deontic statements as part of documentation, or in the statement of requirements. For example, a policy in a hospital could be:

> The accounting department shall not have access to the parts of a patient's record that deal with health history

This requirement will be translated in terms of rules, e.g. if employees of the accounting department attempt to access certain fields in the patient's record, the request will be blocked, or they will be penalized. High-level languages that allow the direct expression of such requirements are becoming available, but eventually these must be translated into rules.

Obligations can be specified in several policy languages for computing systems (notably access control [18] and business-to-business languages [14]), however in this context they don't seem to have the same meaning. In the computing context, to say that a behavior is forbidden simply means that it will not take place. To say that a behavior is compulsory means that it will take place if the conditions are verified. It could be claimed that such obligations or permissions are in fact rules.

Therefore, there is a difference between the meaning of deontic modalities in law and in IT policies, difference that must be resolved before we can use such concepts interchangeably in the two domains.

It is our view that, although modal logics have been used extensively in both Computing and Jurisprudence, the tendency to make them a ubiquitous paradigm should be resisted, because they add a level of complexity while many types of analysis can be done without them.

5. Rules and requirements

We have therefore identified two normative styles:

- The *rule style* (of which examples are the Hammurabi and the firewall style)
- The *requirements style* (of which an example is the Moses style)

This is consistent with the distinction between requirement and implementation in software methodology. The deontic style specifies requirements to be implemented by means of rules, just as software specifications must be implemented by means of programs.

In societal terms, it appears that the rule style is self-sufficient: a society can be run by it. However the enforcement of a requirement seems to depend on the existence of rules that say what will happen if the requirement is broken.

The distinction between 'rules' and 'principles' in legal theory is explored in [25]. This paper makes the point that rules and principles are the extremes of a spectrum, rather than two essentially different normative styles. This is reasonable, however we contend that in a well-structured normative system these two types of norms should be clearly identified and separated, just as requirements and implementation are kept separate in software methodology. The Hammurabi and the Moses codes are very consistent in their use of styles.

There are of course many other normative styles, including styles that have a place between the two identified above, and others whose interpretation and classification could be the subject of endless discussion. However as mentioned we shall schematize and reason in terms of these two styles.

6. Common research topics

Within this framework, there are several research topics that are equally relevant in the areas of law and in the area of policy systems in IT. In these topics, common methods can be used. These are the topics of Completeness, Consistency, and Conflicts.

In law, as in software methodology, questions of completeness and consistency can arise:

- Between rules
- Between requirements
- Between rules and requirements

These questions may be difficult to answer because of logical interrelationships among norms. In an access control system, there may be a rule stating that only executives can access budget information, as well as definitions from which it is possible to deduce that receptionists are not executives, so it will be possible to conclude that receptionists cannot access budget information. This derived rule can be inconsistent with respect to others, or can fill an apparent gap.

6.1. Completeness

Are all cases covered in a law, are all cases covered in a software specification or in a program? What are the 'cases'? In law, conceivably they are all possible social situations. In computing, they are all possible inputs to a system, and rigid type systems are used to limit consideration to certain types of inputs; other inputs will simply be ignored, typically resulting in error messages.

In logic, a system is complete if for any statement P, a proof exists for either P or not P. In normative systems and jurisprudence, there are different and more complicated definitions [1, 3, 4], but one wonders why.

In law, one could at first think that if a situation is not considered, then there is no rule for it, and anything goes. In criminal law, no provision usually means that the situation is tolerated, it is sometimes said that the system contains an implicit *closure norm* to this effect: "nullum crimen sine lege".

To reason more systematically, let us consider the relationships that we have established between rules and requirements.

First of all, consider rules only or requirements only. In programming, we can have a series of tests on a variable of a certain type. If some values of the type are not considered, then the program will go to an explicit or implicit 'otherwise' statement, which plays the role of a closure norm, so incompleteness is impossible.

But normative systems often do not have 'otherwise' clauses, and in the case of law often the data types are not clearly defined. The cases that are not considered remain in some sort of limbo that has generated much literature [1, 3, 4]. The legislator's *intention* comes into consideration, and this can be the result of inductive reasoning. The intention becomes a sort of implicit requirement. Suppose that there is a domain that consists of subdomains some of which are covered by norms, and others that are not, without an explicit norm that covers all subdomains. For example, consider the case where the downtown of a city consists of streets A, B, C, and D. Different rules punish parking on A, B and C with different fines. One can conclude that parking on all downtown streets, except D, is forbidden, or that possibly the system is incomplete.

Other examples are discussed in detail in [1]. Suppose that there are norms for the case where A and B are true, and for the case where *not A* and *not B* are true. What is the norm for the case where only one of A or B is true? The fact that some cases are considered and others are not can point to an implicit requirement that all cases should be considered.

Similar situations of course can arise in the area of software requirements, if these are manually generated and interpreted.

In some domains, it is a requirement that a decision must be always achieved. An example of this situation is provided by firewall systems. A firewall must decide acceptance or rejection for each and every packet. So if there is no rule for a specific packet type, the system must apply some sort of 'default norm' to decide. For example, in Cisco firewalls, if there is no rule for a certain packet, then the packet is refused. In Linux firewalls, the default decision is acceptance. These are the 'closure norms'.

A similar situation exists in inheritance law, where a way must be found to distribute the whole inheritance.

The cases we have discussed make reference to implicit requirements. Of course, incompleteness can also be caused by relating explicit requirements with sets of explicit rules.

For example, the following norm of the Canadian Charter of Rights and Freedoms: 'Everyone has the right to life…' has been used to argue that Canada's law is incomplete because it has no norms to address abortion.

The introduction of new requirements will likely generate incompleteness, which must be filled by the introduction of new rules.

The treatment of incompleteness is very different between legal systems and IT systems. In IT, incompleteness is pathological and the system must be made complete, either by default rules or by intervention of the designer.

It is interesting to note that in legal systems filling incompleteness is often considered to be a role of the judiciary. For this reason, it is sometimes said that legal systems cannot be incomplete, because a judge can always find the rule to apply [3,4]. In IT, one could think of providing a system with meta-rules to solve incompleteness, however, unless these rules are very simple (such as the mentioned closure rule), incompleteness or inconsistency can exist at this meta-level.

6.2. Consistency

In classical logic, a system is consistent if there is no statement A for which it is possible to prove both *A* and *not A*. Different rules can cover the same cases with contradictory effects. This situation of course cannot be confused with the similar case in which two rules can be applied, but their results are compatible (e.g. one norm stipulates a repayment, another stipulates a fine). As for incompleteness, we can identify the following cases:

- Inconsistencies between rules
- Inconsistencies between requirements
- Inconsistencies between rules and requirements

In classical logic, in an inconsistent system anything can be derived, because an inconsistency is false and from false anything can be derived. Therefore, any inconsistency has global implications. However this conclusion is insignificant in practice. If an enterprise database contains an inconsistency, users normally still believe the rest, although the more inconsistencies are found, the more confidence will decrease. If an inconsistency is found in rules for a complex game, players will still play the game according to the remaining rules. Therefore the users of an inconsistent system tend more to isolate the inconsistent part, than to say that since the system is inconsistent it can't be used. And in those cases that are not flagrant, users try to iron out inconsistencies by means of interpretation, i.e. by trying to show that different assertions apply to different cases. In other words, it may be possible to interpret the clauses in such a way that the inconsistency disappears. This is a main occupation for judges and lawyers.

In software methodology, inconsistency among requirements is generating a literature [7, 16], and the solutions proposed are complex.

The case of inconsistencies between rules is explicitly considered in several types of IT systems. In most cases (programs, firewalls) the rules are executed top-down and this will automatically eliminate inconsistency because one of the mutually inconsistent rules won't be reached. However it remains to be seen whether the result corresponds to the intent of the designer. In other systems there are explicit 'combining algorithms' to solve inconsistencies. For example, in the case of the access control language

XACML these are Deny-Overrides, Permit-Overrides, First Applicable, and Only-One Applicable, plus others that can be defined by the user [18].

In Western jurisprudence, some overriding principles have been known for centuries, such as (in their Latin versions): *lex specialis derogat legi generali, lex posterior derogat legi priori, lex superior derogat legi inferiori*, i.e. a law can be overridden by a more specific one, or by a later one, or by one of higher hierarchical position.

Still, there is question of whether the application of the chosen algorithm or principle may betray the intention of the author of the norms, who may not fully understand all existing conflicts and their possible solutions.

In IT, much study has been generated by a particular type of inconsistency, called Feature Interaction. This subject attracted the attention of designers of telephony features, when they realized that the combination of several features led in some cases to problems, because one feature could disrupt the intended effect of another. This could occur on one end of the system, or, worse, between ends. The more general case of this problem can be stated in the following way. In a component-based system, some components may have mutually inconsistent requirements. How can this be detected, can they still be combined? A series of workshops and conferences has been dedicated to this topic, they started in 1992 and the most recent has been [18].

An analysis of inconsistency of norms from the legal and deontic point of view is presented in [5].

It appears then that the most practical solution for inconsistency in normative systems is to report the existing inconsistencies to the designer of the system for human resolution.

6.3. Ontologies

Advanced normative systems use extensive sets of definitions to structure the domain on which they act. Family and inheritance laws are typical examples. Companies have organizational structure that is taken into consideration in company policies. For example, employees can be characterized by roles. We are all familiar with call processing systems that forward calls on specific matters to employees with a certain role. The well-known access control method called RBAC (for Role-Based Access Control) [6] uses roles to determine access rights to databases or other resources.

These definitions form ontologies, which are hierarchical data structures containing attributes for the entities in a certain domain, together with their relationships. Some literature [5] refers to ontologies with the name of *world knowledge*, which they contrast with *normative knowledge.*

The conditions that we have mentioned with relation to ECA systems refer to ontologies.

Ontologies and definitions can act as rule generators: e.g. we can have a norm saying that theft is punished in a certain way, then definitions saying that certain behaviors are theft. The combination of these definitions with norms that use them creates new norms. By using RBAC, access control rules are associated to roles rather than to users. Without RBAC, access rules have to be attached to users, so there have to be many more rules, in addition there have to be mechanisms for attaching and detaching rules from users, as the roles of users change. Similarly, in object-oriented languages such as Java, one can define an array of students, and then on this basis one can create a Java class for each student.

Hence, the use of ontologies can substantially simplify and shorten the expression of rules.

Although some normative systems may not contain explicit definitions or ontologies, in reality every such system depends on such information, which may be externally defined or understood in the social context. Firewalls, for example, depend on implicit ontologies such as the structure of the Internet addressing space, the structure of the systems ports, etc.

Ontologies introduce a third normative style with respect to the two identified in Section 5, one that is orthogonal with respect to those. We could call this *ontological*, or *definitional style*.

The study of ontologies for legal systems is arguably the research area in which Jurisprudence seem to be taking the greatest inspiration from Information Technology [2, 23, 24].

6.4. Conflicts between peers

Conflicts can occur between peers when their requests or policies are incompatible, in fact or potentially. In the case where parties are aiming at an agreement, it may be possible to solve conflicts by a negotiation phase, or by concluding that no agreement is possible. In law, this is the material for arbitrators, judges, lawyers. In computing systems, this is material for the operating system, centralized or distributed. The operating system is the government authority in these systems. The difficulty of this subject is visible in many examples. For example, the deadlock problem in operating systems is computationally unsolvable, in the sense that it is impossible to determine that a deadlock is possible or to prevent it.

Although nowadays there is a lot of confidence in peer-to-peer systems and their ability to solve problems by peer-to-peer agreements, issues such as the feature interaction problem and the deadlock problem show that this confidence is unfounded. Authentication also cannot be done on a pure end-to-end basis. It appears that trusted third parties are necessary to solve these problems, as well as others. Trusted third parties are already used in authentication. Hence, distributed systems will acquire some of the architectural characteristics of legal systems, with their legislators, judges and notaries.

7. Normative systems for electronic societies

Societies are ruled by laws and customs. Those who do not abide by them are punished or emarginated. Electronic societies can span the world and these enforcement methods may not be effective, as we all know by our constant fight against spam and viruses. Similarly, we can get into what appears to be a bona fide electronic agreement with a party, and then we are helpless when we see that the party does something we don’t believe was agreed.

A model for peer-to-peer agreements and electronic societies may be provided by international law, which is constituted mainly of customs and multilateral conventions.

Collaboration of distributed systems can only be achieved if they all use certain common mechanisms. For example, interprocess communication in a set of distributed Java processes depends on all the processes using synchronized methods. Such tacit agreements constitute customs.

In law and society, many customs exist that are respected by all who want to be considered reliable citizen. For example, if Alice lends a book to Bob, Bob is supposed to check back with Alice if she wants to lend it to Carl. However in telephony Bob can automatically forward to Carl a call from Alice without checking with her. So Alice may find herself talking to Carl, although she may have Carl on her incoming call screening list. This is a well-know example of interaction of telephony features that is possible because of violation of a rule that is well understood in society, but not so in telephony.

Unequal agreements with network entities such as Skype, who dictates terms of operation, may be similar to protectorates. Consortia such as Apache, where participants can collaborate in the evolution of the system, are more similar to alliances. In computing systems we can have multi-faceted situations where a user can be simultaneously in many such agreements, again creating the possibility of inconsistencies. People routinely click the 'Accept all conditions' box, happily no one compares all such clauses that have been accepted.

Network entities will associate with other entities they can trust, and this will establish societies of mutual trust. Concepts and modes of operations will be created in these societies, some of which will slowly gain acceptance, thus enlarging the societies. The concept of Web of Trust is an application of this idea.

8. Useful concepts and tools

8.1. Defeasible logic

We have seen that in some systems there are implicit meta-rules by which some rules take the priority. I.e. in firewalls, the rules that come first take the priority. This is not justifiable in logic terms, because order has no importance in logic. Similarly, in legal system all norms are equally valid unless otherwise said.

Defeasible logic is a logic already well-known in AI and philosophy of law. It is a non-monotonic logic first proposed in [17]. It involves three types of propositions:

- Hard rules: these specify that a fact is always a consequence of another: all packets from spammers.com must be refused.
- Defeasible rules: specify that a fact is *typically* a consequence of another: all packets from nuisance.com must be refused
- Defeaters: specify *exceptions* to defeasible rules: packets from luigi@nuisance.com must be accepted.

Therefore, before applying a defeasible rule, it must be checked for defeaters. Defeasible logic provides a framework for specifying exceptions and priorities. Interestingly, it is difficult to find hard rules in nowadays' legal systems, a fact that creates a lot of work for lawyers and judges…

The closure norm can be seen as a defeasible norm. It exists in the system, but can be defeated by any other norm. It applies only if no other norm applies. If defeasible logic is not used, the closure norm can be constructed as the norm that applies when the conjunction of the negation of the premises of all other norms is true, and this conjunction may be very lengthy indeed.

8.2. Theorem Provers and Model Checkers

The commonality of concepts and issues leads to a commonality of automated tools that can be used in this research area.

Theorem provers and model checkers can be used to prove consistency and completeness, although they face computational complexity constraints and are difficult to use. Alloy [8] provides a notation and a tool that seem to ease in part these problems.

Both in IT and in law, such methods face different challenges of scale and of precision. At analysis time, it must be decided on what aspects the analysis should be concentrated in order to make it feasible.

8.3. Tools for Ontologies

It was mentioned above that there is considerable similarity of methods between Jurisprudence and IT in the use of ontologies. In fact, OWL, the Web Ontology Language for the Semantic Web, together with its tools, is a common reference in both areas [2, 11].

9. Conclusions

In the new era of e-commerce and agent societies, IT is encountering some of the problems that have motivated Jurisprudence for thousands of years. Increasingly, the behavior of information systems is gaining legal relevance. On the other hand, IT can contribute to transform these problems by injecting new dimensions, as well as methods and tools for precision and for quick, automatic decisions.

Similarities and relationships can be found in the areas of architecture (the system of legal institutions in Jurisprudence) and norms. In this paper we have concentrated on this second area.

We have argued that there are many principles in common among different types of normative systems, including IT and Jurisprudence. These common principles can be studied in general terms and common methods and tools can be developed. Our classification of normative styles in Section 5 has allowed us to draw several parallels.

Acknowledgment. This research has been funded in part by the Natural Sciences and Engineering Research Council of Canada. I am indebted to Guido Governatori of the University of Queensland for contributing ideas with detailed comments on an earlier version of this paper, and for introducing me to defeasible logic. As well, I am indebted to my PhD student Waël Hassan for many discussions on related topics. I would like to dedicate this paper to the memory of Fritz Paradies, a lawyer and scholar in Frankfurt and Amsterdam, who wrote about normative concepts in programming languages in the 1960s.

References

1. Alchourròn, C.E., Bulygin, E.: *Normative Systems*. Springer, 1971.
2. Casanovas, P., Biasiotti, M.A., Francesconi, F., Sagri, M.T. (Eds): Proc. of the Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT 2007), June 2007.
3. Chiassoni, P.: A tale from two traditions: Civil law, Common law, and legal gaps. Analisi e Diritto, 2006, 51-74. http://www.giuri.unige.it/intro/dipist/digita/filo/testi/analisi_2007/ accessed June 2007.
4. Conte, A.G. : *Saggio sulla completezza degli ordinamenti giuridici*. Giappichelli, 1962.
5. Elhag, A. A. O., Breuker, J. A. P. J., Brouwer, P. W. : On the formal analysis of normative conflicts. Information & Communications Technology Law, 9:3 (2000), 207– 217.
6. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: *Role-based Access Control*. Artech House, 2003.
7. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Transactions on Software Engineering and Methodology, 14(3):277-330, July 2005.
8. Jackson, D.: Software Abstractions. MIT Press, 2006.
9. Jones, A.J.I., Sergot, M.: On the characterisation of law and computer systems: The normative systems perspective. In: *Deontic Logic in Computer Science: Normative System Specification*, J.-J.C. Meyer and R.J. Wieringa (Eds), Wiley, 1993.
10. Kelsen, H.: *General Theory of Law and State*. Harvard University Press, 1945.
11. Lacy, L.W.: *OWL: Representing Information Using The Web Ontology Language*. Trafford, 2005.
12. Lee, A. J., Boyer, J. P., Olson, L. E., and Gunter, C. A.: Defeasible security policy composition for web services. In Proc. of the Fourth ACM Workshop on Formal Methods in Security (Alexandria, Virginia, USA, November 03 - 03, 2006).
13. McIver, W.J.: Software support for multilingual legislative drafting. CIRN Conference and Colloquium, Oct. 2004.
14. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H., Elmagarmid, A.K.: Business-to-business interactions: issues and enabling technologies. The Very Large Data Base Journal (2003) 12: 59–85.
15. Meyer, J.J.C., Wieringa, R.J. (Eds.): *Deontic Logic in Computer Science: Normative System Specification*. Wiley, 1993.
16. Nuseibeh, B., Easterbrook, S., Russo, A.: Making Inconsistency Respectable in Software Development, Journal of Systems and Software, 58(2):171-180, 2001.
17. Nute, D.: Defeasible logic. In *Handbook of logic in artificial intelligence and logic programming, volume 3*: Nonmonotonic reasoning and uncertain reasoning, 353-395. Oxford University Press, 1994.
18. OASIS, Organization for the Advancement of Structured Information Standards.: XACML, eXtensible Access Control Language. www.oasis-open.org/committees/xacml/ accessed May 2007.
19. Paton, N.W. (Ed.): *Active Rules in Database Systems*. Springer, 1999.
20. Pitt. J. (Ed.): *Open Agent Societies: Normative Specifications in Multi-Agent Systems*. Wiley, 2004.

21. Reiff-Marganiec, S., Ryan, M.D.: *Feature Interactions in Telecommunications and Software Systems VIII*, IOS Press, 2005.
22. Sartor, G.: *Legal Reasoning: A Cognitive Approach to the Law*. Springer, 2005.
23. Valente, A.: *Legal knowledge engineering: A modeling approach.* IOS Press, 1995.
24. van Kralingen, R.W., van den Herik, H.J., Prins, J.E.J., Sergot, M., Zeleznikow J. (eds.): Legal Knowledge Based Systems: Foundations of legal knowledge systems. Proc. of Jurix 2006, IOS Press, 2006.
25. Verheij, B., Hage, J.C., van den Herik, H. J.: An integrated view on rules and principles. Artificial Intelligence and Law, Vol. 6 (1998), No. 1, 3-26