DR-CONTRACT: An Architecture for e-Contracts in Defeasible Logic

Guido Governatori* and Duy Hoang Pham
NICTA, Queensland Research Laboratory, Brisbane, Australia
email: {guido.governatori,duyhoang.pham}@nicta.com.au
* Corresponding author

Abstract: We introduce the DR-CONTRACT architecture to represent and reason on e-Contracts. The architecture extends the DR-device architecture by a deontic defeasible logic of violation. We motivate the choice for the logic and we show how to extend *RuleML* to capture the notions relevant to describe e-contracts for a monitoring perspective in Defeasible Logic.

Keywords: Defeasible Deontic Logic, Violations, e-Contract.

Reference to this paper should be made as follows: Guido Governatori and Duy Hoang Pham (2009) 'DR-CONTRACT: An Architecture for e-Contracts in Defeasible Logic', *Int. J. Business Process Integration and Management*, Vol. X, No. Y, pp.W–Z.

Biographical notes: Guido Governatori received his Ph.D. in computer science and law at the University of Bologna in 1997. Since then he has held academic and research positions at Imperial College, Griffith University, Queensland University of Technology, the University of Queensland, and NICTA. He has published more than 160 scientific papers in logic, artificial intelligence, and database and information systems. His current research interests include modal and nonclassical logics, defeasible reasoning and its application to normative reasoning and e-commerce, agent systems, and business process modeling for regulatory compliance. He is a member of the editorial board of Artificial Intelligence and Law.

Duy Hoang Pham is a lecturer at the Faculty of Information Technology, Posts and Telecommunications Institute of Technology in Vietnam since 1998. In 2000, he was awarded Master of Technology in Computing (specialisied in Intelligent Systems) at RMIT University, Australia. From 2005, he carried out PhD research at the School of Information Technology and Electrical Engineering, the University of Queensland and at NICTA, Queensland Research Alboratory. His research is sponsored by the Ministry of Education and Training of Vietnam and NICTA.

1 Introduction

Business contracts are mutual agreements between two or more parties engaging in various types of economic exchanges and transactions. They are used to specify the obligations, permissions and prohibitions that the signatories should be hold responsible to and to state the actions or penalties that may be taken in the case when any of the stated agreements are not being met.

We will focus on the monitoring of contract execution and performance: contract monitoring is a process whereby activities of the parties listed in the contract are governed by the clauses of the contract, so that the correspondence of the activities listed in the contract can be monitored and violations acted upon. In order to monitor the execution and performance of a contract we need a precise representation of the 'content' of the contract to perform the required actions at the required time.

The clauses of a contract are usually expressed in a codified or specialised natural language, e.g., legal English. At times this natural language is, by its own nature, imprecise and ambiguous. However, if we want to monitor the execution and performance of a contract, ambiguities must be avoided or at least the conflicts arising from them resolved. A further issue is that often the clauses in a contract show some mutual inter-dependencies and it might not be evident how to disentangle such relationships. To implement

an automated monitoring system all the above issues must be addressed.

To deal with some of these issues we propose a formal representation of contracts. A language for specifying contracts needs to be formal, in the sense that its syntax and its semantics should be precisely defined. This ensures that the protocols and strategies can be interpreted unambiguously (both by machines and human beings) and that they are both predictable and explainable. In addition, a formal foundation is a prerequisite for verification or validation purposes. One of the main benefits of this approach is that we can use formal methods to reason with and about the clauses of a contract. In particular we can

- analyse the expected behaviour of the signatories in a precise way, and
- identify and make evident the mutual relationships among various clauses in a contract.

Secondly, a language for contracts should be conceptual. This, following the well-known *Conceptualization Principle* of Griethuysen (1982), effectively means that the language should allow their users to focus only and exclusively on aspects related to the content of a contract, without having to deal with any aspects related to their implementation.

Every contract contains provisions about the obligations, permissions, entitlements and others mutual normative positions the signatories of the contract subscribe to. Therefore a formal language intended to represent contracts should provide notions closely related to the above concepts.

A contract can be viewed as a legal document consisting of a finite set of articles, where each article consists of finite set of clauses. In general it is possible to distinguish two types of clauses:

1. definitional clauses, which define relevant concepts occurring in the contract;
2. normative clauses, which regulate the actions of the parties for contract performance, and include deontic modalities such as obligations, permissions and prohibitions.

For example the following fragment of a contract of service taken from Governatori (2005) are definitional clauses

> 3.1 A "Premium Customer" is a customer who has spent more that $10000 in goods.
>
> 3.2 Service marked as "special order" are subject to a 5% surcharge. Premium customers are exempt from special order surcharge.

while

> 5.2 The (Supplier) shall on receipt of a purchase order for (Services) make them available within one day.

and

> 5.3 If for any reason the conditions stated in 4.1 or 4.2 are not met the (Purchaser) is entitled to charge the (Supplier) the rate of $100 for each hour the (Service) is not delivered.

are normative clauses. The above fragment should make it it clear that there is a deep conceptual difference between Clauses 3.1 and 3.2 on one side, and Clauses 5.2 and 5.3 on the other. The first two clauses are factual/definitional clauses describing states of affairs, defining notions in the conceptual space of the contract. For example clause 3.1 defines the meaning of "Premium Customer" for the contract, and Clause 3.2 gives a recipe to compute the price of services. On the other hand Clauses 5.2 and 5.3 state the (expected) legal behaviour of the parties involved in the transaction. In addition there is a difference between Clause 5.2 and Clause 5.3. Clause 5.2 determines an obligation for one of the parties; on the other hand Clause 5.3 establishes a permission. Hence, according to our previous discussion about the functionalities of the representation formalism, a logic meant to capture the semantics of contracts has to account for such issues. For contracts we must be able to distinguish whether the non-compliance with a clause of a contract constitutes a breech of the contract or not (for normative clauses) or when it is just outside the scope of the contract (for definitional clauses).

Since the seminal work by Lee (1988) Deontic Logic has been regarded as one on the most prominent paradigms to formalise contracts. Governatori (2005) further motivates on the need of deontic logic to capture the semantics of contracts and the reasons to choose it over other formalisms.

Clause 3.2 points out another feature. Contract languages should account for exceptions. In addition, given the normative nature of contracts, exceptions can be open ended, that is, it is not possible to give a complete list of all possible exception to a condition. This means that we have to work in an environment where conclusions are defeasible, i.e., it is possible to retract conclusions when new pieces of information become available.

From a logical perspective every clause of a contract can be understood as a rule where we have the conditions of applicability of the clause and the expected behaviour. Thus we have that we can represent a contract by a set of rules, and, as we have already argued, these rules are non-monotonic. Thus we need a formalism that is able to reason within this kind of scenario. Our choice here is Defeasible Logic (we will motivate this choice in section).

Finally Clause 5.3 highlights an important aspect of contracts: contracts often contain provisions about obligations/permissions arising in response to violations. Standard Deontic Logic is not very well suited to deal with violations. Many formalisms have devised to obviate some problems of violations in deontic logic. In this paper we will take a particular approach to deal with violation that can be easily combined with the other component we have outlined here.

The paper is organised as follows: in Section we present

the logic on which the DR-CONTRACT architecture is based. Then in Section we explain the extension of *RuleML* corresponding to the logic of the previous section, and we establish a mapping between the two languages. Then, in Section we discuss the system architecture of the DR-CONTRACT framework. Finally we relate our work to similar approaches and we give some insights about future developments in Section .

2 Defeasible Deontic Logic of Violation

For a proper representation of contracts and to be able to reason with and about them we have to combine and integrate logics for various essential component of contracts. In particular we will use the Defeasible Deontic Logic of Violation (DDLV) proposed by Governatori (2005). This logic combines deontic notions with defeasibility and violations. More precisely DDLV is obtained from the combination of three logical components: Defeasible Logic, deontic concepts, and a fragment of a logic to deal with normative violations. Before presenting the logic we will discuss the reasons why such notions are necessary for the representation of contracts.

Grosof (2004) advances Courteous Logic Programming (CLP) as the inferential engine for business contracts represented in *RuleML*. Here, instead, we propose Defeasible Logic (DL) as the inferential mechanism for *RuleML*. In fact, CLP is just a notational variant of one of the many logics in the family proposed by Antoniou et al. (2000b,a) (see Antoniou et al. (2000c) for the relationships between DL and CLP, and Antoniou et al. (2006) for the relationships between DL and Logic Programming in general). Accordingly, it may be possible to integrate the extensions we develop in the rest of the paper within a CLP framework. Antoniou et al. (2000a) demonstrate that DL is be a flexible non-monotonic formalism able to capture different and sometimes incompatible facets of non-monotonic reasoning, and efficient and powerful implementations have been proposed (for example, Antoniou et al. (2000b); Maher et al. (2001); Bassiliades et al. (2006)). The primary use of DL in the present context is aimed at the resolution of conflicts that might arise from the clauses of a contract; in addition, according to Governatori et al. (2004) DL encompasses other existing formalisms proposed in the AI & Law field, and Governatori and Rotolo (2004); Governatori et al. (2005); Padmanabhan et al. (2006); Governatori and Rotolo (2008b) show that DL is suitable for extensions with modal and deontic operators.

DL analyses the conditions laid down by each rule in the contract, identifies the possible conflicts that may be triggered and uses priorities, defined over the rules, to eventually solve a conflict. A defeasible theory contains here four different kinds of knowledge: facts, strict rules, defeasible rules, and a superiority relation.

Facts are indisputable statements, for example, "the price of the spam filter is \$50". Facts are represented by predicates

$$Price(SpamFilter, 50).$$

Strict rules are rules in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "A 'Premium Customer' is a customer who has spent \$10000 on goods", formally:

$$TotalExpense(X, 10000) \rightarrow PremiumCustomer(X).$$

Defeasible rules are rules that can be defeated by contrary evidence. An example of such a rule is "Premium Customer are entitled to a 5% discount":

$$PremiumCustomer(X) \Rightarrow Discount(X).$$

The idea is that if we know that someone is a Premium Customer, then we may conclude that she is entitled to a discount *unless there is other evidence suggesting that she may not be* (for example if she buys a good in promotion).

The *superiority relation* among rules is used to define priorities among them, that is, where one rule may override the conclusion of another rule. For example, given the defeasible rules

$$r : PremiumCustomer(X) \Rightarrow Discount(X)$$
$$r' : SpecialOrder(X) \Rightarrow \neg Discount(X)$$

which contradict one another, no conclusive decision can be made about whether a Premium Customer who has placed a special order is entitled to the 5% discount. But if we introduce a superiority relation $>$ with $r' > r$, then we can indeed conclude that special orders are not subject to discount.

We now give a short informal presentation of how conclusions are drawn in Defeasible Logic. A conclusion p can be derived if there is a rule whose conclusion is p, whose prerequisites (antecedent) have either already been proved or given in the case at hand (i.e. facts), and any stronger rule whose conclusion is $\neg p$ has prerequisites that fail to be derived. In other words, a conclusion p is derivable when:

- p is a fact; or
- there is an applicable strict or defeasible rule for p, and either
  - all the rules for $\neg p$ are discarded (i.e., are proved to be not applicable) or
  - every applicable rule for $\neg p$ is weaker than an applicable strict[1] or defeasible rule for p.

Antoniou et al. (2001); Governatori (2005) offer full presentations of Defeasible Logic.

We illustrate the inferential mechanism of Defeasible Logic with the help of an example. Let us assume we have a theory containing the following rules:

$$r_1 : PremiumCustomer(X) \Rightarrow Discount(X)$$
$$r_2 : SpecialOrder(X) \Rightarrow \neg Discount(X)$$
$$r_3 : Promotion(X) \Rightarrow \neg Discount(X)$$

[1] Notice that a strict rule can be defeated only when its antecedent is defeasibly provable.

where the superiority relation is thus defined: $r_3 > r_1$ and $r_1 > r_2$. The theory states that services in promotion are not discounted, and so are special orders with the exception of special orders placed by premium customers, who are normally entitled to a discount.

In a scenario where all we have is that we received a special order, then we can conclude that the price has to be calculated without a discount since rule r_1 is not applicable (we do not know whether the customer is a premium customer or not). In case the special order is received from a special customer for a service not in promotion, we can derive that the customer is entitled to a discount. Indeed rule r_1 is now applicable and it is stronger than rule r_2, and r_3, which is stronger than r_2 is not applicable (i.e., the service is not in promotion).

The next step is to integrate deontic logic in defeasible logic. To this end we follow the idea presented by Governatori and Rotolo (2004). In the context of contract we introduced the directed deontic operators $O_{s,b}$ and $P_{s,b}$. Thus, for example the expression $O_{s,b}A$ means that A is obligatory such that s is the subject of such an obligation and b is its beneficiary; similarly for $P_{s,b}$, where $P_{s,b}A$ means that s is permitted to do A in the interest of b. In this way it is possible to express rules like the following

$$PurchaseOrder \Rightarrow O_{Supplier,Purchaser}DeliverWithin1Day$$

that encodes Clause 5.2 of the contract presented above.

Finally, let us sketch how to incorporate a logic for dealing with normative violations within the framework we have described so far. A violation occurs when an obligation is disattended, thus $\neg A$ is a violation of the obligation OA. However, a violation of an obligation does not imply the cancellation of such an obligation. This makes often difficult to characterise the idea of violation in many formalisms for defeasible reasoning (see, among others van der Torre and Tan (1997)). We will take and adapt some intuitions we developed fully by Governatori and Rotolo (2002, 2006). To reason on violations we have to represent contrary-to-duties (CTDs) or reparational obligations. As is well-known, these last are in force only when normative violations occur and are meant to "repair" violations of primary obligations. In the spirit of Governatori and Rotolo (2002, 2006) we introduce the non-classical connective $\otimes$, whose interpretation is such that $OA \otimes OB$ is read as "OB is the reparation of the violation of OA". The connective $\otimes$ permits to combine primary and CTD obligations into unique regulations. The operator $\otimes$ is such that $\neg\neg A \equiv A$ for any formula A and enjoys the properties of associativity, duplication and contraction. For the purposes of this paper, it is sufficient to define the following rule for introducing $\otimes$:[2]

$$\frac{\Gamma \Rightarrow O_{s,b}A \otimes (\bigotimes_{i=1}^{n} O_{s,b}B_i) \otimes O_{s,b}C \qquad \Delta, \neg B_1, \ldots, \neg B_n \Rightarrow \mathbf{X}_{s,b}D}{\Gamma, \Delta \Rightarrow O_{s,b}A \otimes (\bigotimes_{i=1}^{n} O_{s,b}B_i) \otimes \mathbf{X}_{s,b}D} \quad (1)$$

[2]The $\otimes$ is allowed only in the head of defeasible rules. Governatori (2005) fully motivates this design choice.

where $\mathbf{X}$ denotes an obligation or a permission. In this last case, we will impose that D is an atom. Since the minor premise states that $\mathbf{X}_{s,b}D$ is a reparation for $O_{s,b}B_n$, i.e., the last literal in the sequence $\bigotimes_{i=1}^{n} O_{s,b}B_i$, we can attach $\mathbf{X}_{s,b}D$ to such sequence. In other words, this rule permits to combine into a unique regulation the two premises.

Suppose the theory includes

$$r : Invoice \Rightarrow O_{s,b}PayWithin7Days$$
$$r' : \neg PayWithin7Days \Rightarrow O_{s,b}PayWithInterest.$$

From these rules we obtain

$$r'' : Invoice \Rightarrow O_{s,b}PayWithin7Days \otimes O_{s,b}PayWithInterest.$$

As soon as we applied ($\otimes$I) as much as possible, we have to drop all redundant rules. This can be done by means of the notion of subsumption:

Definition 1 *Let $r_1 = \Gamma \Rightarrow A \otimes B \otimes C$ and $r_2 = \Delta \Rightarrow D$ be two rules, where $A = \bigotimes_{i=1}^{m} O_{s_i,b_i}A_i$, $B = \bigotimes_{i=1}^{n} O_{s_i,b_i}B_i$ and $C = \bigotimes_{i=1}^{p} \mathbf{X}_{s_i,b_i}C_i$. Then r_1* subsumes *r_2 iff*

1. *$\Gamma = \Delta$ and $D = A$; or*
2. *$\Gamma \cup \{\neg A_1, \ldots, \neg A_m\} = \Delta$ and $D = B$; or*
3. *$\Gamma \cup \{\neg B_1, \ldots, \neg B_n\} = \Delta$ and $D = A \otimes \bigotimes_{i=0}^{k \leq p} \mathbf{X}_{s_i,b_i}C_i$.*

The idea behind this definition is that the normative content of r_2 is fully included in r_1. Thus r_2 does not add anything new to the system and it can be safely discarded. In the example above, we can drop rule r, whose normative content is included in r''.

Formally a *conclusion* in DDLV is a tagged literal and can have one of the following forms:

- $+\Delta q$ to mean that the literal q is definitely provable (i.e., using only facts and strict rules),
- $-\Delta q$ when q is not definitely provable,
- $+\partial q$, whenever q is defeasibly provable, and
- $-\partial q$ to mean that we have proved that q is not defeasibly provable.

Provability is based on the concept of a *derivation*. A derivation is a finite sequence $P = (P(1), \ldots, P(n))$ of tagged literals satisfying four conditions (which correspond to inference rules for each of the four kinds of conclusion). Here we will give only the conditions for $+\Delta$ and $+\partial q$. $P(1..i)$ denotes the initial part of the sequence P of length i:

The inference rule for $\pm\Delta$ are just those for forward chaining of strict rules, thus they corresponds to detachment or Modus Ponens for $+\Delta$ and a full search that modus ponens cannot be applied for $-\Delta$.

To accommodate the new connective ($\otimes$) in DDLV we have to revise the inference mechanism of Defeasible Logic. The first thing we have to note is that now a defeasible rule

can be used to derive different conclusions. For example given the rule

$$r : A \Rightarrow O_{s,b}B \otimes O_{s,b}C \tag{2}$$

we can use it to derive $O_{s,b}B$ if we have A, but if we know A and $\neg B$ then the same rule supports the conclusion $O_{s,b}C$.

With $R[c_i = q]$ we denote the set of rules where the head of the rule is $\otimes_{j=1}^{n} c_j$ where for some i, $1 \leq i \leq n$, $c_i = q$. For example, given the rule r in (2), $r \in R[c_1 = O_{s,b}B]$ and $r \in R[c_2 = O_{s,b}C]$. Given an obligation $O_{s,b}A$, we use $\overline{O_{s,b}A}$ to denote the complement of A, i.e., $\sim A$.

We are now ready to give the proof condition for $+\partial$.

$+\partial$: If $P(i+1) = +\partial q$ then either
(1) $+\Delta q \in P(1..i)$ or
(2) (2.1) $\exists r \in R[c_i = q]$
(2.1.1) $\forall a \in A(r) : +\partial a \in P(1..i)$ and
(2.1.2) $\forall i' < i, \exists a = \overline{c_{i'}} : +\partial a \in P(1..i)$
(2.2) $-\Delta{\sim}q \in P(1..i)$ and
(2.3) $\forall s \in R[c_j = {\sim}q]$ either
(2.3.1) $\exists a \in A(s) : -\partial a \in P(1..i)$ or
(2.3.2) $\exists j' < j, \forall c_{j'} - \partial\overline{c_{j'}} \in P(1..i)$ or
(2.3.3) $\exists t \in R_{sd}[q]$ such that
$\forall a \in A(t) : +\partial a \in P(1..i)$
$\forall k' < k, +\partial\overline{c_{k'}} \in P(1..i)$ and $t > s$.

The above condition is very similar to the same condition for basic defeasible logic given by Antoniou et al. (2001). The main differences account for the $\otimes$ connective. What we have to ensure is that reparations of violations are in force when we try to prove them. For example if we want to prove $O_{s,b}C$ given the rule $r : A \Rightarrow O_{s,b}B \otimes O_{s,b}C$, we must show that we are able to prove A, and that the primary obligation B has been violated. In other words we have already proved $\neg B$ or any other formula incompatible with B (Clause 2.1.2). A similar explanation holds true for Clause 2.3.2 where we want to show that a rule does not support an attack on the intended conclusion.

Conflicts often arises in contracts. What we have to determine is whether we have genuine conflicts, i.e., the contracts is in some way flawed or whether we have *prima-facie* conflicts. A prima-facie conflict is an apparent conflict that can be resolved when we consider it in the context where it occurs and if we add more information the conflict disappears. For example let us consider the following two rules:

$$r : \mathit{PremiumCustomer} \vdash O_s\mathit{Discount}$$
$$r' : \mathit{SpecialOrder} \vdash O_s\neg\mathit{Discount}$$

saying that Premium Customers are entitled to a discount (r), but there is no discount for goods bought with a special order (r'). Is a Premium customer entitled to a discount when she places a special order? If we only have the two rules above there is no way to solve the conflict just using the contract and there is the need of a domain expert to advise the knowledge engineer about what to do in such case. The logic can only point out that there is a conflict in the contract. On the other hand, if we have an additional provision

$$r'' : \mathit{PremiumCustomer}, \neg\mathit{Discount} \vdash O_s\mathit{Rebate}$$

Specifying that if for some reasons a premium customer did not received a discount then the customer is entitled to a rebate on the next order, then it is possible to solve the conflict, because the contract allows a violation of rule r to be amended by r'', using the merging mechanism of rule (1).

The following rule is devised for making explicit conflicting norms (contradictory norms) within the system:

$$\frac{r : \Gamma \Rightarrow A \qquad r' : \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow \bot} \tag{3}$$

where[3]

1. for any formula C, $\{C, \neg C\} \not\subseteq \Gamma \cup \Delta$; and
2. it is not the case that either $r > r'$ or $r' > r$; and either
3. $A = O_{s,b}C$ and $B = \neg O_{s,b}C$; or
4. $A = \neg O_{s,b}C$ and $B = O_{s,b}C$; or
5. if $A = O_{s,b}C$ and $B = O_{s,b}\neg C$, then
   - there is no rule $\Gamma' \vdash X$ such that either $\neg C \in \Gamma'$, or $X = O_{s,b}C \otimes D$ and $\Gamma' \subseteq \Gamma \cup \Delta$; and
   - there is no rules $\Delta' \vdash Y$ such that either $C \in \Delta'$, or $Y = O_{s,b}\neg C \otimes D$ and $\Delta' \subseteq \Gamma \cup \Delta$.

The meaning of the first condition is that there is a situation where both rules are applicable, this means that the states of affairs/preconditions they require are consistent. The second condition ensures that the two rules have the same strength, if one of them is stronger than the other, we use the superiority relation to solve the conflict. For conditions 3–5 we have to distinguish two different types of conflicts. For conditions 3and 4, the conflict is that for something we have at the same time an obligation and there is no obligation for it, i.e., $O_{s,b}C$ and $\neg O_{s,b}C$[4]

For condition 4 the intuition is that given two rule, we have a conflict if the normative content of the two rules is opposite, such that none of them can be repaired. The eventual reparations cannot happen, since the would require inconsistent states of affairs.

Once conflicts have been detected there are several ways to deal with them. The first thing to do is to determine whether we have a *prima-facie* conflict or a genuine conflict. As we have seen we have a conflict when we have two rules with opposite conclusions. Thus a possible way

[3]For the application of this rules, we consider that all formulas $P_{s,b}A$ are transformed into $\neg O_{s,b}\neg A$.

[4]Alternatively we can say that something is at the same time forbidden and permitted, given the equivalences between the deontic operators, i.e., $F_{s,b}C$ (forbidden C) is equivalent to $O_{s,b}\neg C$, and $P_{s,b}C$ (permitted C) is equivalent to $\neg O_{s,b}\neg C$.

to solve the conflict is to create a superiority relation over the rules and to use it do "defeat" the weaker rule, or the designer of the contract can use the information given by the rule labelled as inconsistent to revise the contract to avoid the problem.

3 Normal Forms and Canonical Forms

In the previous section we have presented the formal machinery of DDLV. Given a formal representation of a contract we can use the logic to reason with the conditions of a contract. For example we can use it at run time to determine whether a particular situation complies with the contract. Similarly the inference engine provided by DDLV can run dry tests at design time to verify correctness of the representation of a contract (i.e., that the conclusions obtained from a scenario are those expected by the designer of the contract.)

In this section we examine how the formalism can be used to analyse contracts and to reason about them so that ambiguities in a contract can be identified.

It is possible that two contract domain engineers come up with different representations for the same contract. This might also be the case when one designer formalises a (part of) contract at different times. DDLV can facilitate the comparison of two different versions of the same contract to determine whether they are equivalent. DDLV can also be used to ensure consistency between draft of a contracts during the negotiation phase of the contract: we compare the drafts of the contract of the negotiating parties.

We introduce transformations of an DDLV representation of a contract to produce normal form of the same (NDDLV). A *normal form* is a representation of a contract based on an DDLV specification containing all contract conditions that can generated/derived from the given DDLV specification. The purpose of a normal form is to "clean up" the DDLV representation of a contract, that is to identify formal loopholes, deadlocks and inconsistencies in it, and to make hidden conditions explicit.

As discuss before it is possible to have different versions of a contract. For example,

Normal forms can be beneficial in comparing two versions of a contract for equivalence and compatibility. In case we have different DDLV representations of a contract, and their respective normal forms are not equivalent, then it may be useful to consolidate them into a unifying version that integrates the conditions expressed in the normal forms. We call the resulting representation the *canonical form* of the contract (CDDLV).

Since canonical forms are complete and hence contain all conditions of a contract they can be mapped to an executable representation, aimed at the implementation and monitoring of the same.

Notice that there can be many normal forms for a contract, but there is only one canonical form, since normal forms are the expansions of formal specifications of (potentially a part of) contract. The idea is that a normal form is the closure under some logical operations of a fragment of a contract, while the canonical form is the closure of all fragments of a contract (fragments can overlap).

Figure 1 illustrates a scenario where there are two equivalent (formal) versions of the contract DDLV_1 and DDLV_2. The two versions are equivalent since they produce the same normal form (NDDLV_1). On the other side DDLV_3 corresponds to a normal form that does not coincide with NDDLV_1. Thus we can compare and integrate the two normal forms to produce the canonical form of the contract CDDLV, which in turn is mapped to an executable program, or to a strorage or interchange format (e.g., in *RuleML*).

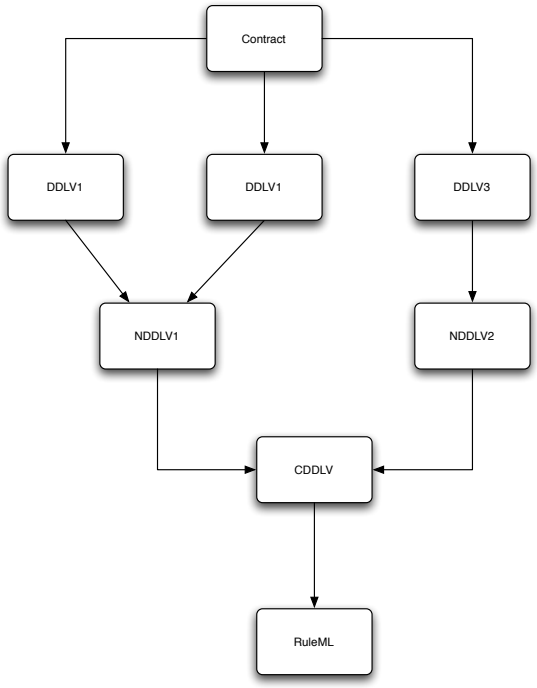

Figure 1: DDLV Normalisation Process

The normalisation process consists of the following three steps:

1. Starting from a formal representation of the explicit clauses of a contract we generate all the implicit conditions that can be derived from the contract by applying the merging mechanism of DDLV, rule (1).

2. We can clean the resulting representation of the contract by throwing away all redundant rules according to the notion of subsumption, Definition 1.

3. Finally we use the conflict identification rule to label and detect conflicts, using 3).

In general the process at step 2 must be done several times in the appropriate order as described above. The normal form of a set of rules in DDLV is the fixed-point of the above constructions. A contract contains only finitely many rules and each rule has finitely many elements. In

addition Governatori and Rotolo (2006) to show that the operation on which the construction is defined is monotonic, thus according to standard set theory results the fixed-point exists and it is unique. However, we have to be careful since merging first and doing subsumption after produces different results from the opposite order (i.e., subsumption first and merging after), or by interleaving the two operations.

If there is only one normal form of a contract then the normal form coincides with the canonical form of the contract. In case there are multiple normal forms of a contract, for example if the contract has been built in a modular way from several (sub-) contract templates (e.g.. based on the idea by Hoffner and Field (2005)), we have to combine the normal forms to check for their completeness and mutual consistency. This means that we have to union the sets of rules from each normal form and to repeat the fixed-point construction of step 2, and then to identify the eventual conflicts. After these operations we obtain the canonical form of the contract. A domain expert can use the canonical form to check that the representation of a contract covers all aspects of the contract, and, in case of conflicts, she suggests which interpretation is the more faithful to the intent of the contract, and she can point out features included in the contract but missing in its formal representation.

Another application of the normalisation procedure is that it can be used in the negotiation phase of a contract life-cycle. Given a draft of a contract, to be negotiated among parties, the parties involved in the negotiation provide their DDLV representations of the contract. Then the parties exchange their DDLV versions of the contract, and run the normalisation procedure. If the DDLV versions converge into a unique normal form NDDLV (and so to the canonical form of the contract or CDDLV), then each party has the option to agree on the canonical form or to propose amendments in case the party believes that some features of the contract are not included in the canonical form. If all parties agree then the canonical form can be taken as the agree (formal) interpretation of the parties. In case some parties propose extensions, the extended DDLV formalisations can be shared by the parties, and the whole process repeated. In case we have multiple normal forms of the contract, then this means that there are conflicts among the interpretations put forward by the parties. The conflicts can be identified using the mechanism presented in the previous section and the parties can then negotiate solutions to the conflicts.[5]

4 Contracts in RuleML

In order to integrate the the DR-CONTRACT engine with Semantic Web technology we decided to use RuleML (2009) as an open and vendor neutral XML/RDF syntax for contracts. We tried to re-use as many features of standard RuleML syntax as possible. However, since some notions essential for the representation of contracts are not present in standard RuleML we have created our DR-CONTRACT DTD (Figure 2).[6]

```
<!ELEMENT Atom       (Not?,Rel,(Ind|Var)*)>
<!ELEMENT Not        (Rel,(Ind|Var)*)>
<!ELEMENT Rel        (#PCDATA)>
<!ELEMENT Var        (#PCDATA)>
<!ELEMENT Ind        (#PCDATA)>
<!ELEMENT Fact       (Atom)>
<!ELEMENT Imp        ((Head,Body)|(Body|Head))>
<!ATTLIST Imp        label ID #REQUIRED
                     strength (strict|
                               defeasible) #REQUIRED>
<!ELEMENT Body       (And)>
<!ELEMENT And        (Atom|Obligation|Permission)*>
<!ELEMENT Head       (Atom|Obligation|
                      Permission|Behaviour)>
<!ELEMENT Behaviour  ((Obligation)+,Permission?)>
<!ELEMENT Obligation (Not?,Rel,(Ind|Var)*)>
<!ATTLIST Obligation subject IDREF #REQUIRED
                     beneficiary IDREF #REQUIRED>
<!ELEMENT Permission (Not?,Rel,(Ind|Var)*)>
<!ATTLIST Permission subject IDREF #REQUIRED
                     beneficiary IDREF #REQUIRED>
```

Figure 2: DR-CONTRACT Basic DTD

The main limitations of RuleML is that it does not support modalities and it is unable to deal with violations. The DR-CONTRACT RuleML DTD takes two different types of literals: unmodalised predicates and modalised literals. Thus to appropriately represent the deontic notions of obligation and permission we introduce two new elements `<Obligation>` and `<Permission>`, which are intended to replace `<Atom>` in the conclusion of normative rules. In addition deontic elements can be used in the body of derivation rules. Hence we have to extend the definition of `<And>` and `<Head>`. In this way it is possible to distinguish from brute fact and normative facts. As we have already argued this is essential if one wants to use RuleML to represent business contracts.

The elements `<Var>` and `<Ind>` are, respectively, placeholders for individual variables to be instantiated by ground values when the rules are applied and individual constants. Individual constants can be just simple names or URIs referring to the appropriate individuals. `<Rel>` is the element that contains the name of the predicate. `<Not>` is intended to represent classical negation. Thus its meaning is that the atom it negates is not the case (or the proposition represented by the atom is false and consequently the

[5] It is not the scope of the present paper to address contract negotiation, The only aspect we want to remark is that the methodology presented here can be used in the contract negotiation phase. For models of contract negotiation see, among others, Reeves et al. (2002); Rittgen (2008); Bacarin et al. (2008).

[6] Although the current version of RuleML (Version 0.91) is based on XML Schema, here due to space limitation and for ease of presentation, we will give the XML grammar using simplified DTD definitions.

proposition the element represents is true). *RuleML* contains two types of negation, classical negation and negation as failure Wagner (2002); Boley et al. (2001). However, Antoniou et al. (2000c) show that negation as failure can be simulated by other means in Defeasible Logic, so we do not include it in our syntax.

RuleML provides facilities for many types of rule. However, we believe that the distinction has a pragmatic flavour more than a conceptual one. In this paper we are interested in the logical and computational aspects of the rules, thus we decided to focus only on derivation rules `<Imp>`.

Derivation rules allow the derivation of information from existing rules. They are able to capture concepts not stored explicitly in the existing information. For example, a customer is labelled as a "Premium" customer when he buys $10000 worth of goods. As such, the rule here states that the customer must have spent $10000 on goods, thus deriving the information here that the customer is a "Premium" customer. A derivation rule has an attribute `strength` whose value ranges over `strict` and `defeasible` and it denotes the type of rule to be associated to it when computed in defeasible logic.

A derivation rule has two immediate sub-elements, *Condition* (`<Body>`) and *Conclusion* (`<Head>`); the latter being either an atomic predicate formula or a sequence of obligations, and the former a conjunction of formulas, meaning that derivation rules consist of one more conditions and a conclusion.

The ability to deal with violations and the obligations arising in response to them is one of the key features in the representation of business contracts. To this end the conclusion of a derivation rule corresponding to a normative rule is a `<Behaviour>` element, defined as a sequence of `<Obligation>` and `<Permission>` elements with the constraints that the sequence contains at most one `<Permission>` element, and this element is the last of the sequence. This construction is meant to simulate the behaviour of $\otimes$.

As we have alluded to in the previous section RuleML provides a semantically neutral syntax for rules and different types of rules can be reduced to other types and rules in RuleML can be mapped to native rules in other formalism. For the relationships between RuleML and Defeasible Logic we will translate derivation rules (`<Imp>`s) into rules in Defeasible Logic specifications. In this perspective a derivation rule

```
<Imp label="r" strength="defeasible">
  <Body>...</Body>
  <Head>
    <Behaviour>
      <Obligation>A1</Obligation>
      ...
      <Deontic>An</Deontic>
    </Behaviour>
  </Head>
</Imp>
```

is transformed into a defeasible rule

$$r : \texttt{body} \Rightarrow OA_1 \otimes \cdots \otimes \mathbf{X}A_n$$

where $\mathbf{X}$ is the translation of the `<Deontic>` (meta) element.

We give now an example of a rule based on the following contract clause

> 6.1 The payment terms shall be in full upon receipt of invoice. Interest shall be charged at 5 % on accounts not paid within 7 days of the invoice date.

```
<Imp label="6.1"
     strenght="defeasible">
   <Body>
      <And>
         <Atom><Rel>Invoice</Rel>
               <Var>InvoiceDate</Var>
               <Var>Amount</Var>
         </Atom>
      </And>
   </Body>
   <Head>
      <Behaviour>
         <Obligation subject="Purchaser"
                     beneficiary="Supplier">
               <Rel>PayInFullWithin7Days</Rel>
               <Var>InvoiceDate</Var>
               <Var>Amount</Var>
         </Obligation>
         <Obligation subject="Purchaser"
                     beneficiary="Supplier">
               <Rel>PayWithInterest</Rel>
               <Var>Amount * 1.05</Var>
         </Obligation>
      </Behaviour>
   </Head>
</Imp>
```

```
<!ELEMENT And       (Atom|Obligation|
                     Permission|Violation)*>
<!ELEMENT Violation EMPTY>
<!ATTLIST Violation rule IDREF #REQUIRED>
<!ELEMENT Behaviour ((Obligation+,Reparation)|
                     (Obligation*,Permission?))>
<!ELEMENT Reparation EMPTY>
<!ATTLIST Reparation penalty IDREF #REQUIRED>
<!ELEMENT Penalty   ((Obligation+,Reparation)|
                     (Obligation*,Permission?))>
<!ATTLIST Penalty   label ID #REQUIRED>
```

Figure 3: DR-CONTRACT Extended DTD

The new deontic tags in the DR-CONTRACT extended DTD in Figure 3 –`<Reparation>`, `<Penalty>` and `<Violation>`– do not increase the expressive power of the language but are included as convenient shortcuts. It is possible to express a violation explicitly by saying that a particular rule is triggered in response to a violation (i.e., when an obligation is not fulfilled). However, it can be convenient to have facilities to represent violations directly –just look at the formulation of Clause 5.3. In general a violation can be one of the conditions that trigger the application of a rule. Accordingly a `<Violation>` element can be included in the body of a rule. A violation cannot subsist without a rule that is violated by it. Hence

the attribute `rule` is a reference to the rule that has been violated. Many contract languages, for example, the languages proposed by Grosof and Poon (2003) and Milosevic et al. (2004), contain similar constructions. The activation of such constructions/processes requires the generation of a violation event/literal. On the contrary our approach does not require it. All we have to do is to check for a sequence of literals joined with the $\otimes$ operator where the initial part of the sequence is not satisfied.

A `<Violation>` occurs in the body of rule and the `rule` attribute refers to the violated rule. Every `<Violation>` element can be replaced by the conjunction of the elements in the `<Body>` of the violated rule, i.e., the rule the `rule` attribute refers to, plus the negation of the un-modalised elements of the elements in the `<Head>` of the violated rule.

```
<Imp label="v">
  <body>B1</body>
  <head>
    <Behaviour>
      <Obligation>A1</Obligation>
      ...
      <Obligation>An</Obligation>
    </Behaviour>
  </head>
</Imp>
<Imp label="r">
  <body>
    <And>
      B2
      <Violation rule="v"/>
    </And>
  </body>
  <head>
    <Behaviour>
      <Obligation>C1</Obligation>
      ...
      <Deontic>Cm</Deontic>
    </Behaviour>
  </head>
</Imp>
```

From the above *RuleML* code we generate two rules in DDLV, namely

$$\begin{aligned} v &: B_1 \Rightarrow OA_1 \otimes \cdots \otimes OA_n, \\ r &: B_1, B_2, \neg A_1, \ldots, \neg A_n \Rightarrow OC_1 \otimes \cdots \otimes \mathbf{X}C_m. \end{aligned}$$

Eventually the two rules can be combined via the schema (1) in

$$vr : B_1, B_2 \Rightarrow OA_1 \otimes \cdots \otimes OA_n \otimes OC_1 \otimes \cdots \otimes \mathbf{X}C_m.$$

In some cases one might have recurrent general penalties and it may be convenient to state them once and refer back to them when they are called. To deal with this case we introduce two additional elements `<Reparation>` and `<Penalty>`. A `<Reparation>` element is just an empty element with a reference to a `<Penalty>` element that can occur only after an obligation in a `<Behaviour>` element, where a `<Penalty>` element is a premiseless rule with a normative head that is triggered only when its corresponding violations are raised.

For example given the following fragment of a contract

```
<Imp label='r'>
  <body>...</body>
  <head>
    <Behaviour>
      <Obligation>A1</Obligation>
       ...
      <Obligation>An</obligation>
      <Reparation penalty="p"/>
    </Behaviour>
  </head>
</Imp>
 <Penalty label="p">
    <Obligation>B1</Obligation>
    ...
    <Deontic>Bm</Deontic>
 </Penalty>
```

the rule corresponding to it is

$$r : \texttt{body} \Rightarrow OA_1 \otimes \cdots \otimes OA_n \otimes OB_1 \otimes \cdots \otimes \mathbf{X}B_m.$$

5 DR-CONTRACT System Architecture

The system architecture of DR-CONTRACT is inspired by the system architecture of the family of DR-DEVICE applications developped by Skylogiannis et al. (2005); Bassiliades et al. (2006) and consists of four main modules (see Figure 4):

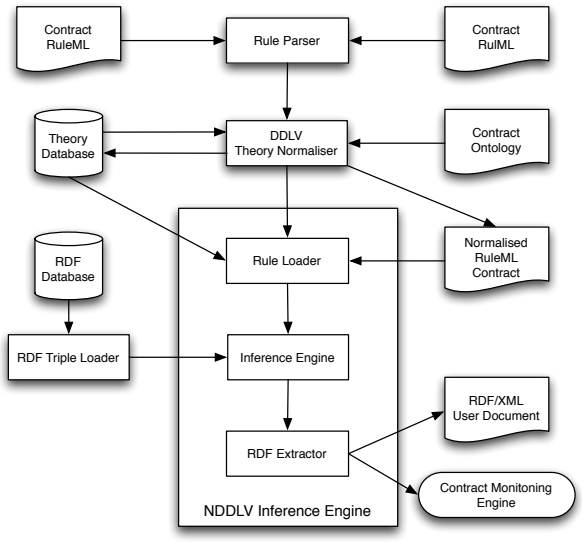

Figure 4: DR-CONTRACT System Architecture

1. A Rule Parser to transform a DR-CONTRACT compliant document (a contract) into a theory to be passed to the next module. The parser is based on the XML processor and it is rather similar in nature to the LogicLoader module of the DR-Device family applications by Skylogiannis et al. (2005); Bassiliades et al. (2006).

2. A DDLV normaliser. The normaliser takes as input a set of DDLV theories (obtained from the previous step) and an RDF ontology. The ontology is

used to ensure that alignment of the predicates and other resources used by the *RuleML* compliant contracts. Then it iteratively merges rules in the theory according to the inference rule 1 and then removes rules subsumed by a more general rule according to Definition 1. It repeats the cycle till it reaches the fixed-point of such a construction (Governatori and Rotolo (2006) prove that it allways exists and that it is unique). Once a theory has been normalised the normal/canonical form is saved to a repository (for faster loading in successive calls), and the normalised theory NDDLV is passed to the DDLV engine. In addition the normaliser applies a transformation that removes superiority relation by compiling it into new rules (the technique used here is similar to the transformation proposed by Antoniou et al. (2001)).

3. The RDF loader downloads/queries the input documents, including their schemata, and it translates the RDF descriptions into fact objects according to the RDF-NDDLV translation schema based on the DR-CONTRACT DTD.

4. The NDDLV inference engine consists of two components:

   - The *Rule Loader* compiles the rules in a NDDLV theory in objects. We distinguish two types of objects: Rules and Literals or atoms. Each rule object has associated to it a list of (pointers to) modal literals (corresponding to head of the rule) and a set of (pointers to) modal literals implemented as an hash table. Each atom object has associated to it four hash tables: the first with pointers to the rules where the atom occurs positively in the head, the second with pointers to the rules where the atom occurs negatively in the head, the third with pointers to the rules where the atom occurs positively in the body and the last with pointers where the atom occurs negatively in the body.
   - The *Inference Engine* is based on an extension of the *Delores* algorithm/implementation proposed by Maher et al. (2001) as a computational model of Basic Defeasible Logic. In turn:
     - It asserts each fact (as an atom) as a conclusion and removes the atom from the rules where the atom occurs positively in the body, and it "deactivates" the rules where the atom occurs negatively in the body. The complement of the literal is removed from the head of rules where it does not occur as first element. The atom is then removed from the list of atoms.
     - It scans the list of rules for rules where the body is empty. It takes the first element of the head and searches for rule where the negation of the atom is the first element. If there are no such rules then, the atom is appended to the list of facts, and removed from the rules
     - It repeats the first step.
     - The algorithm terminates when one of the two steps fails. On termination the algorithm outputs the set of conclusions.[7]

5. Finally the conclusions are exported either to the user or to a monitoring contract facility such as BCL by Milosevic et al. (2004); Linington et al. (2004) as an RDF/XML document through an RDF extractor. Governatori and Milosevic (2006) show how to map FCL specifications to BCL specifications. The mapping can be used to interface our framework with a BCL contract monitoring implementation.

6 Conclusion and Related Works

In this paper we have presented a system architecture for a Semantic Web based system for reasoning about contracts. The architecture is inspired by the system architecture of the DR-DEVICE family of applications. The main differences between our approach and the DR-DEVICE is in the use of an extended variant of Defeasible Logic. The extensions are in the use of modal operator and a non classical operator for violations. The same difference applies for the SweetDeal approach by Grosof (2004); Grosof and Poon (2003). We have also argued that the extension with modal (deontic) operators is not only conceptually sound but also necessary to capture the semantics of contracts. In the same way the implementation of the inference engine is an extension of the algorithm used by the Delores defeasible logic engine by Maher et al. (2001) to cope with deontic operators and the $\otimes$ operator.

Strano et al. (2008) propose a rule based notation for the specification of executable contracts. The language includes deontic operators, and it is equipped with facilities to handle violations. The main difference with our work is that violations are divided into two classes business violations and technical failures. While this offers a pragmatically interesting feature, conceptually the distinction does not add to the expressive power of the language. This can easily be done in our approach. All we have to do is to insert an additional literal/predicate in the antecedent of the rule for eventually triggering a reparation.

Alberti et al. (2008) present a framework, called $\mathcal{SCIFF}$, for the representation of business contracts and the formal verification of the resulting specification based on abductive logic programming. One of the main aim of the work is to determine whether a logic program, encoding

[7] Governatori et al. (2006b),Governatori and Rotolo (2008b) proved that the algorithm runs in linear time. Each atom/literal in a theory is processed exactly once and every time we have to scan the set of rules, thus the complexity of the above algorithm is $O(|L| * |R|)$, where L is the set of distinct modal literals and R is the set of rules.

a contract, can reach a goal, where the goal is the objective of a contract, what a contract has to achieve. In case a goal cannot be reach from a given situation, $\mathcal{SCIFF}$ tries to abduce some facts that are needed to achieve the goal. The logic programming environment permits to verify the correctness and other properties of a contract. However, the semantics of the abductive logic programs used to model contracts is able to simulate simple deontic operators, but not violations.

The handling of temporal aspects is a very delicate matter in contract monitoring. The current architecture does not cover temporal reasoning. However, Governatori et al. (2005) proposes an extension of Defeasible Logic that can represent and reason with temporalised normative positions. In particular the framework offers facilities to initiate and terminate obligations, permissions, prohibitions and other complex normative positions. We have planned to study how to efficiently incorporate such features in our Deontic Defeasible Logic of Violations.

Currently we have implemented prototypes of the RuleParser based on the ARP parser of Jena (McBride (2001)), the rule normaliser and the inference engine Java. Experimental results show that the implementation of the inference engine is able to deal with some of the benchmark theories of Maher et al. (2001) with theories in some case with over 1,000,000 rules.

We also plan to integrate the framework with the VDR-Device by Bassiliades et al. (2005) to provide a user-friendly graphical *RuleML* editor, recommended by the *RuleML* initiative, and supporting defeasible rule. However, the editor has to be extended to incorporate functionalities to deal with the deontic operators required for the representation of contracts.

Several authors propose the use of workflow and business process management technology to implement e-contract. Governatori et al. (2006a) show how to check the compliance of a business process (modelled in BPMN) using FCL. Given existing mapping between BPMN and BPEL, and the proposed compliance checking mechanism it is possible to monitor the performance of contracts, implemented as BPEL processes, and using DR-CONTRACT to check that the runt time execution of the process implementing the contract is conform to the conditions specified in the contract.

While FCL was initially developed for modelling and reasoning with contracts, the features it presents are general enough to cover a wide class of normative specifications. Following the seminal work by Governatori et al. (2006a), FCL has been proposed to check the compliance of generic processes (not only processes corresponding to contracts). The DDLV normaliser module and the NDDLV inference engine can be use together with the algorithms proposed by Governatori et al. (2008) and Governatori and Rotolo (2008a) to check the compliance of business processes.

Acknowledgements

The paper is an extended and revised version of Governatori and Pham (2005a,b).

We would like to thank Antonino Rotolo and Zoran Milosevic for their fruitful comments on previous versions of this work. Thanks are also due to the CoALa05 anonymous referees for their valuable criticisms.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

Adi, A., Stoutenburg, S., and Tabet, S., editors (2005). *Rules and Rule Markup Languages for the Semantic Web, First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005, Proceedings*, volume 3791 of *Lecture Notes in Computer Science*. Springer.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., and Torroni, P. (2008). Expressing and veryfing business contract with abductive logic programming. *International Journal of Electronic Commerce*, 12(4):9–38.

Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2000a). A flexible framework for defeasible logics. In *Proc. American National Conference on Artificial Intelligence (AAAI-2000)*, pages 401–405, Menlo Park, CA. AAAI/MIT Press.

Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2001). Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287.

Antoniou, G., Billington, D., Governatori, G., and Maher, M. J. (2006). Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming*, 6(6):703–735.

Antoniou, G., Billington, D., Governatori, G., Maher, M. J., and Rock, A. (2000b). A family of defeasible reasoning logics and its implementation. In Horn, W., editor, *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence*, pages 459–463, Amsterdam. IOS Press.

Antoniou, G., Maher, M. J., and Billington, D. (2000c). Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming*, 41(1):45–57.

Bacarin, E., Madeira, E. R., and Medeiros, C. B. (2008). Contract e-negotiation in agricultural supply chains. *International Journal of Electronic Commerce*, 12(4):71–97.

Bassiliades, N., Antoniou, G., and Vlahavas, I. (2006). A defeasible logic reasoner for the Semantic Web. *International Journal on Semantic Web and Information Systems*, 2:1–41.

Bassiliades, N., Kontopoulos, E., and Antoniou, G. (2005). A visual environment for developing defeasible rule bases for the semantic web. In Adi et al. (2005), pages 172–186.

Boley, H., Tabet, S., and Wagner, G. (2001). Design rationale for ruleml: A markup language for semantic web rules. In Cruz, I. F., Decker, S., Euzenat, J., and McGuinness, D. L., editors, *Proceedings of SWWS'01, The first Semantic Web Working Symposium*, pages 381–401.

Governatori, G. (2005). Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216.

Governatori, G., Hoffmann, J., Sadiq, S., and Weber, I. (2008). Detecting regulatory compliance for business process models through semantic annotations. In Ardagna, D., editor, *BPM 2008 Workshops*, volume 7 of *LNBIP*, pages 5–17. Springer.

Governatori, G., Maher, M. J., Billington, D., and Antoniou, G. (2004). Argumentation semantics for defeasible logics. *Journal of Logic and Computation*, 14(5):675–702.

Governatori, G. and Milosevic, Z. (2006). A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(4):659–685.

Governatori, G., Milosevic, Z., and Sadiq, S. (2006a). Compliance checking between business processes and business contracts. In Hung, P. C. K., editor, *10th International Enterprise Distributed Object Computing Conference (EDOC 2006)*, pages 221–232. IEEE Computing Society.

Governatori, G. and Pham, D. H. (2005a). DR-CONTRACT: An architecture for e-contracts in defeasible logic. In Bartolini, C., Governatori, G., and Milosevic, Z., editors, *2nd EDOC Workshop on Contract Architectures and Languages (CoALA 2005)*. IEEE Digital Library. Published on CD.

Governatori, G. and Pham, D. H. (2005b). A semantic web based architecture for e-contracts in defeasible logic. In Adi et al. (2005), pages 172–186.

Governatori, G. and Rotolo, A. (2002). A Gentzen system for reasoning with contrary-to-duty obligations. a preliminary study. In Jones, A. J. and Horty, J., editors, Δ*eon'02*, pages 97–116, London. Imperial College.

Governatori, G. and Rotolo, A. (2004). Defeasible logic: Agency, intention and obligation. In Lomuscio, A. and Nute, D., editors, *Deontic Logic in Computer Science*, number 3065 in LNAI, pages 114–128, Berlin. Springer.

Governatori, G. and Rotolo, A. (2006). Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215.

Governatori, G. and Rotolo, A. (2008a). An algorithm for business process compliance. In Francesconi, E., Sartor, G., and Tiscornia, D., editors, *Legal Knowledge and Information Systems*, volume 189 of *Frontieres in Artificial Intelligence and Applications*, pages 186–191. IOS Press.

Governatori, G. and Rotolo, A. (2008b). BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Journal of Autonomous Agents and Multi Agent Systems*, 17(1):36–69.

Governatori, G., Rotolo, A., and Padmanabhan, V. (2006b). The cost of social agents. In Stone, P. and Weiss, G., editors, *5th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 513–520, New York. ACM Press.

Governatori, G., Rotolo, A., and Sartor, G. (2005). Temporalised normative positions in defeasible logic. In Gardner, A., editor, *10th International Conference on Artificial Intelligence and Law (ICAIL05)*, pages 25–34. ACM Press.

Griethuysen, J. v., editor (1982). *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036.

Grosof, B. N. (2004). Representing e-commerce rules via situated courteous logic programs in RuleML. *Electronic Commerce Research and Applications*, 3(1):2–20.

Grosof, B. N. and Poon, T. C. (2003). SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the twelfth international conference on World Wide Web*, pages 340–349. ACM Press.

Hoffner, Y. and Field, S. (2005). Transforming agreements into contracts. *International Journal of Cooperative Information Systems*, 14(2-3):217–244.

Lee, R. M. (1988). A logic model for electronic contracting. *Decision Support Systems*, 4:27–44.

Linington, P. F., Milosevic, Z., Cole, J. B., Gibson, S., Kulkarni, S., and Neal, S. (2004). A unified behavioural model and a contract language for extended enterprise. *Data & Knowledge Engineering*, 51(1):5–29.

Maher, M. J., Rock, A., Antoniou, G., Billignton, D., and Miller, T. (2001). Efficient defeasible reasoning systems. *International Journal of Artificial Intelligence Tools*, 10(4):483–501.

McBride, B. (2001). Jena: Implementing the RDF model and syntax specification. In *Proc 2nd Int. Workshop on The Semantic Web*.

Milosevic, Z., Gibson, S., Linington, P. F., Cole, J., and Kulkarni, S. (2004). On design and implementation of a contract monitoring facility. In Benatallah, B., editor, *First IEEE International Workshop on on Electronic Contracts*, pages 62–70. IEEE Press.

Padmanabhan, V., Governatori, G., Sadiq, S., Colomb, R. M., and Rotolo, A. (2006). Process modelling: The deontic way. In Stumptner, M., Hartmann, S., and Kiyoki, Y., editors, *Conceptual Modelling 2006. Pro-*

ceedings of the Thirds Asia-Pacific Conference on Conceptual Modelling (APCCM2006), number 53 in CRPIT, pages 75–84, Sydney. Australian Computer Science Communications.

Reeves, D. M., Wellman, M. P., and Grosof, B. N. (2002). Automated negotiation from declarative contract descriptions. *Computational Intelligence*, 18(4):482–500.

Rittgen, P. (2008). A contract-based architecutre for business networks. *International Journal of Electronic Commerce*, 12(4):115–145.

RuleML (2009). RuleML. The Rule Markup Initiative.

Skylogiannis, T., Antoniou, G., Bassiliades, N., and Governatori, G. (2005). DR-NEGOTIATE - a system for automated agent negotiation with defeasible logic-based strategies. In *2005 IEEE International Conference on e-Technology, e-Commerce, and e-Services*, pages 44–49. IEEE Computer Society.

Strano, M., Molina-Jiménez, C., and Shrivastava, S. K. (2008). A rule-based notation to specify executable electronic contracts. In Bassiliades, N., Governatori, G., and Paschke, A., editors, *RuleML*, volume 5321 of *Lecture Notes in Computer Science*, pages 81–88. Springer.

van der Torre, L. and Tan, Y.-H. (1997). The many faces of defeasibility. In Nute, D., editor, *Defeasible Deontic Logic*, pages 79–121. Kluwer, Dordrecht.

Wagner, G. (2002). How to design a general rule markup language. In *Proceedings of XML Technology for the Semantic Web (XSW 2002)*, volume 14 of *LNI*, pages 19–37. GI.