International Journal of Cooperative Information Systems

REPRESENTING BUSINESS CONTRACTS IN *RuleML*

GUIDO GOVERNATORI
School of Information Technology and Electrical Engineering,
The University of Queensland,
Brisbane, QLD 4072, Australia
Email: guido@itee.uq.edu.au

Received (16th November 2004)
Revised (26th February 2005)

This paper presents an approach for the specification and implementation of translating contracts from a human-oriented form into an executable representation for monitoring. This will be done in the setting of *RuleML*. The task of monitoring contract execution and performance requires a logical account of deontic and defeasible aspects of legal language; currently such aspects are not covered by *RuleML*; accordingly we show how to extend it to cover such notions. From its logical form, the contract will be thus transformed into a machine readable rule notation and eventually implemented as executable semantics via any mark-up languages depending on the client's preference, for contract monitoring purposes.

Keywords: Defeasible Logic, Deontic Logic, Logic of Violation, RuleML, Business Contracts

1. Background and Motivation

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 held responsible for and to state the actions or penalties that may be taken when any of the stated agreements are not being met.

Given the increasing efforts by organisations to carry out their business via the Internet, it is crucial to model contracts in terms of workflows, such that all relevant tasks of contracts can be described as elements of business processes, where business processes are constrained by business rules, statements or policies listed in business contracts or other legal documents that are used by organisations to run the activities, to provide an understanding of how a business operates, and to direct the behaviour of the organisation.

Business rules are typically expressed implicitly in documents and they are also hidden in many programs across clients, applications and database tiers of today's business information systems, which make it even harder to monitor the expected behaviour of the policies.

Business rules are usually applicable at two levels: the business domain and the operational level of an information system. In the business domain, business rules are usually described in the form of natural language expressions. However, this form cannot be applied to information systems; hence these statements of business rules must be made operational by transforming them into a declarative (rule) language. The need to formalise business rules explicitly has thus become increasingly essential with the growing reliance on online business exchanges.

This paper focuses on transforming business contract rules from natural language into a machine readable and executable form. In particular, our aim is to implement the contract in a way that allows explicit monitoring of rules by the computer for any case of violation. Based on a given contract scenario, the contract, being in its human-oriented form will be analysed and represented in a logical form using Deontic and Defeasible Logic. The contract will be transformed from its logical form into a machine readable rule notation, based on *RuleML*, and implemented as executable semantics.

In addition the use of logic modelling techniques is beneficial for reasoning about contracts in various ways. Here we outline some possible application areas where the formal representation of a contract can prove useful. In particular we distinguish between benefits for drafting contracts, and benefits for understanding and the application of contracts.

Drafting contracts can be supported in the following ways:

1. *Anomaly detection:* Formal methods can be used to detect anomalies such as inconsistency, incompleteness and circularity. Such anomalies can be detected either by static analysis or by the performance of the proof theory.
2. *Hypothetical reasoning:* It is possible to investigate the effects of changes to clauses of an entire contract. This is possible because a contract is represented as executable specifications.
3. *Debugging:* In many cases we know what the answer to a specific query should be, yet a contract in its current form leads to a different answer. Debugging suggests changes to the contract which will have as an effect the desired outcome. In our project, debugging can be carried out along the lines of "declarative debugging"[28].

Regarding the *understanding and application of contracts*, formal systems have the following advantages. These advantages are important, for example, for "naive users/subjects of contracts" who are regulated but do not wish to study the contract in detail.

4. *Decision support:* It is possible to run a specific case with a given contract to obtain the expected output. For example, given the formal representation of a contract, a user can interrogate the system to determine whether a course of action will result in a breach of the contract before the user commits herself to that course of action.

5. *Explanation:* When an output is given, there is also a reasoning chain explaining this response. This can be most useful in, say, help desks.
6. *Monitoring:* A contract, implemented as executable semantics, can be transformed in a workflow specification for the automated execution of the business processes defined by the clauses of the contract. To this end it is important that all implicit conditions of the contract are made explicit.

The paper is organised as follow. In Section 2 we introduce a simple contract comprising most of the salient features typically present in contracts. The contract will be used to motivate and illustrate how to use the logical framework and *RuleML* in this context. Then in Section 3 we will outline the logical framework required to represent contracts. Section 4 introduces and motivates *RuleML*. In Section 5 we discuss the relationships between Defeasible Logic and *RuleML* before showing (Section 6) how to extend *RuleML* with tags corresponding to the normative notions relevant for contracts, and how to use the logical framework for reasoning with and about contracts (Section 7). We conclude in Section 8 with some final remarks and suggestions for future extensions of this work.

2. A Sample Contract

A contract is a declarative act jointly performed by all parties whose status is going to be changed by the declaration they are performing. Such joint declarations are usually performed by combining two acts, the first of which is called offer and the second acceptance[11]. As is well-known, this general perspective has been widely adopted in the e-commerce domain, and indeed communicative aspects are crucial in scenarios such as the contract net protocol[30,11]. However, we are not interested here in modelling of negotiation and establishment of contracts. Rather, 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 legally, so that the correspondence of the activities listed in the contract can be monitored and violations acted upon.

This paper is based on the analysis of the following sample contract.

CONTRACT OF SERVICES

This Deed of Agreement is entered into as of the Effective Data identified below.

BETWEEN
ABC Company (To be known as the Purchaser)

AND
ISP Plus (To be known as the Supplier)

WHEREAS (Purchaser) desires to enter into an agreement to purchase from (Supplier) Application Server (To be known as (Goods) in this Agreement).

NOW IT IS HEREBY AGREED that (Supplier) and (Purchaser)

shall enter into an agreement subject to the following terms and conditions:

1 Definitions and Interpretations

2.1 Price is a reference to the currency of the Australia unless otherwise stated.
2.2 This agreement is governed by Australia law and the parties hereby agree to submit to the jurisdiction of the Courts of the Queensland with respect to this agreement.

2 Commencement and Completion

3.1 The commencement date is scheduled as January 30, 2002.
3.2 The completion date is scheduled as January 30, 2003.

3 Price Policy

3.1 A "Premium Customer" is a customer who has spent more that $10000 in goods. Premium Customers are entitled a 5% discount on new orders.
3.2 Goods marked as "special order" are subject to a 5% surcharge. Premium customers are exempt from special order surcharge.
3.3 The 5% discount for premium customers does not apply for goods in promotion.

4 Purchase Orders

4.1 The (Purchaser) shall follow the (Supplier) price lists at http://supplier.com/catalog1.html.
4.2 The (Purchaser) shall present (Supplier) with a purchase order for the provision of (Goods) within 7 days of the commencement date.

5 Service Delivery

5.1 The (Supplier) shall ensure that the (Goods) are available to the (Purchaser) under Quality of Service Agreement (http://supplier/qos1.htm). (Goods) that do not conform to the Quality of Service Agreement shall be replaced by the (Supplier) within 3 days from the notification by the (Purchaser), otherwise the (Supplier) shall refund the (Purchaser) and pay the (Purchaser) a penalty of $1000.
5.2 The (Supplier) shall on receipt of a purchase order for (Goods) make them available within 1 days.
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 (Goods) are not delivered.

6 Payment

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. The prices shall be as stated in the sales order unless otherwise agreed in writing by the (Supplier).
6.2 Payments are to be sent electronically, and are to be performed under standards and guidelines outlined in PayPal.

7 Termination NOT SHOWN TO SAVE SPACE

8 Disputes NOT SHOWN TO SAVE SPACE

SIGNATURES

In a nutshell, the items covered within this contract are: (a) the roles of the parties; (b) the period of the contract (the times at which the contract is in force); (c) the nature of consideration (what is given or received), e.g., actions or items; (d) the obligations and permissions associated with each role, expressed in terms of criteria over the considerations, e.g., quality, quantity, cost and time; (e) the domain of the contract (which determines the rules under which the validity, correctness, and enforcement of the contract will operate). 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. Following our sample, there are basically 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.

3. The Logical Framework

In this section we sketch the basics of the logical apparatus used in the paper. Basically, we will combine three logical components: Defeasible Logic, deontic concepts, and a fragment of a logic intended to deal with normative violations.

RuleML [32] is used here to make explicit all conditions of contracts in a machine readable language, which, in turn, is transformed into executable code. *RuleML* provides a way of expressing business rules as modular, stand-alone units[35,36]. It also possesses the ability to resolve conflicts using priorities and override predicates[8,36].

3.1. *Defeasible Logic*

Courteous logic programming (CLP) has been advanced as the inferential engine for business contracts represented in *RuleML* [18,17,19]. Here, instead, we propose Defeasible Logic (DL) as the inferential mechanism for *RuleML*. In fact, CLP is just one of the many variants of DL[6]. Over the years DL proved to be a flexible nonmonotonic formalism able to capture different and sometimes incompatible facets of nonmonotonic reasoning[4], and efficient and powerful implementations have been proposed[26,7,17]. 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. DL analyses the conditions laid down by each rule in the contract, identifies the possible conflicts that may be triggered and uses the priorities defined over the rules to eventually resolve a conflict. In addition DL encompasses other existing formalisms for normative reasoning developed in the AI & Law field[13] such as Prakken and Sartor's[31] and Loui and Simari's[33], and recent work shows that DL is suitable for extensions with modal and deontic operators[15].

In the rest of the section we first give an informal presentation of Defeasible Logic and then we present it formally. Finally we illustrate how to use it with the help of an example.

3.1.1. *A Defeasible Logic Primer*

A defeasible theory contains five different kinds of knowledge: facts, strict rules, defeasible rules, defeaters, 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, defeasible rules and defeaters are represented, respectively, by expressions of the form $A_1, \ldots, A_n \rightarrow B$, $A_1, \ldots, A_n \Rightarrow B$ and $A_1, \ldots, A_n \rightsquigarrow B$, where $A_1, \ldots, A_n$ is a possibly empty set of prerequisites and B is the conclusion of the rule. We only consider rules that are essentially propositional. Rules containing free variables are interpreted as the set of their ground instances.

Strict rules are rules in the classical sense: whenever the premises are indisputable then so is the conclusion. Thus they can be used for definitional clauses. An example of a strict rule is "A 'Premium Customer' is a customer who has spent \$10000 on goods" (Clause 3.1, part 1):

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

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" (Clause 3.1, part 2):

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

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, Clause 3.3).

Defeaters are a special kind of rules. They are used to prevent conclusions not to support them. For example:

$$SpecialOrder(X), PremiumCustomer(X) \rightsquigarrow \neg Surcharge(X).$$

This rule (Clause 3.2, part 2) states that premium customers placing special orders might be exempt from the special order surcharge. This rule can prevent the derivation of a "surcharge" conclusion. On the other hand it cannot be used to support a "not surcharge" conclusion. It is worth noting that *RuleML* does not support defeaters: this is not a limitation since defeaters do not augment the expressive power of a theory; in fact defeaters can be simulated by defeasible rules[5]. Moreover it is not clear to us whether defeaters correspond to "natural" and "intuitive" clauses in contracts. In the rest of the paper we will not make use of defeaters.

Defeasible Logic is a "skeptical" non-monotonic logic, meaning that it does not support contradictory conclusions. Instead Defeasible Logic seeks to resolve conflicts. In cases where there is some support for concluding A but also support for concluding $\neg A$, Defeasible Logic does not conclude either of them (thus the name "skeptical"). If the support for A has priority over the support for $\neg A$ then A is concluded. This means that the designer of a Defeasible Logic theory (a contract) has to identify pairs of *incompatible literals*. Two literals are said to be incompatible if they cannot both hold at the same time, which essentially means that one of the literals implies the negation of the other. This means that for every literal A, A and its negation $\neg A$ are incompatible, but there are other cases. For example, let us suppose we have the predicates *PremiumCustomer* and *BasicCustomer*. If we know that, according to the interpretation of the business rules behind the contract we want to model they cannot be true at the same time for one and the same individual; then we can specify that *PremiumCustomer* and *BasicCustomer* are incompatible which each other. This means that $PremiumCustomer(a)$ and $BasicCustomer(a)$ cannot both be true for the same customer a. Another example of conflicting literals regards the price of goods, where for each literal $Price(X, Y)$ the literals incompatible with it are all literals $Price(X, Z)$, where $Z \neq Y$. This construction ensures that for every good X, the price to be paid for it is unique (even if this price can be calculated from the advertised price and the eventual discount and surcharge, and distinct instances of it can be different).

Having identified conflicting literals, with the aid of an inference tool we can then detect conflicting (defeasible) rules, i.e., rules such that the literals appearing in the conclusions are incompatible. As we have alluded to above, no conclusion can be drawn from conflicting rules in defeasible logic unless these rules are prioritised. The *superiority relation* among rules is used to define priorities among rules, that is, where one rule may override the conclusion of another rule. For example, given the defeasible rules

$$r : PremiumCustomer(X) \Rightarrow Discount(X)$$

and

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

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[a] or defeasible rule for p.

The formal definitions of derivations in Defeasible Logic are in the next section; the reader not interested in the technical details can skip directly to Section 3.1.3 where we provide a comprehensive example of how to use Defeasible Logic.

3.1.2. *A Formal Presentation of Defeasible Logic*

Now we present defeasible logics formally. A *rule* r consists of its *antecedents* (or *body*) $A(r)$ which is a finite set of literals, an arrow, and its *consequent* (or *head*) $C(r)$ which is a literal. There are three kinds of arrows, $\rightarrow$, $\Rightarrow$ and $\rightsquigarrow$ which correspond, respectively, to strict rules, defeasible rules and defeaters. Where the body of a rule is empty or consists of one formula only set notation may be omitted in examples.

For each literal p we define the set of *p-Incompatible literals* ($\mathcal{I}(p)$), that is, the set of literals that cannot hold when p does. We stipulate that the negation of a literal is always complementary to the literal. Let us consider again the predicates *PremiumCustomer* and *BasicCustomer*. If according to the business rules those two predicate cannot be both true for one and the same customer, then we define, for any constant a, $\mathcal{I}(PremiumCustomer(a)) = \{\neg PremiumCustomer(a), BasicCustomer(a)\}$.

Given a set R of rules, we denote the set of all strict rules in R by R_s, the set of strict and defeasible rules in R by R_{sd}, the set of defeasible rules in R by R_d, and the set of defeaters in R by R_{dft}. $R[q]$ denotes the set of rules in R with consequent q, and $R[\mathcal{I}(q)]$ denotes the set of rules in R whose consequent is in $\mathcal{I}(q)$.

A *defeasible theory* D is a structure

$$D = (F, R, <, \mathcal{I})$$

where F is a finite set of facts, R is a finite set of rules, $<$ is a binary relation over R, and $\mathcal{I}$ is a function mapping a literal to a set of literals (the set of literals incompatible with it).

A *conclusion* in DL 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.

[a]Notice that a strict rule can be defeated only when its antecedent is 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:

$+\Delta$: If $P(i+1) = +\Delta q$ then
(1) $q \in F$ or
(2) $\exists r \in R_s[q]\ \forall a \in A(r) : +\Delta a \in P(1..i)$.

This definition describes forward chaining of strict rules. For a literal q to be definitely provable we need to find a strict rule with head q ($r \in R_s[q]$), of which all antecedents have been definitely proved previously ($\forall a \in A(r) : +\Delta a \in P(1..i)$.).

$-\Delta$: If $P(i+1) = -\Delta q$ then
$\forall r \in R_s[q] \exists a \in A(r) : -\Delta a \in P(1..i)$.

To establish that q cannot be proven definitely we must establish that for every strict rule with head q there is at least one antecedent which has been shown to be non-provable.

Now we turn to the more complex case of defeasible provability. Before giving its formal definition we provide the idea behind such a notion. A defeasible proof of a literal p consists of three phases. In the first phase either a strict or defeasible rule is put forth in order to support a conclusion p; then we consider an attack on this conclusion using the rules for its negation $\neg p$. The attack fails if each rule for $\neg p$ is either discarded (it is possible to prove that part of the antecedent is not defeasibly provable) or if we can provide a stronger counterattack, that is, if there is an applicable strict or defeasible rule stronger than the rule attacking p. It is worth noting that defeaters cannot be used in the last phase.

$+\partial$: If $P(i+1) = +\partial q$ then either
(1) $+\Delta q \in P(1..i)$ or
(2) (2.1) $\exists r \in R_{sd}[q] \forall a \in A(r) : +\partial a \in P(1..i)$ and
(2.2) $\forall p \in \mathcal{I}(q) - \Delta p \in P(1..i)$ and
(2.3) $\forall s \in R[\mathcal{I}(q)]$ either
(2.3.1) $\exists a \in A(s) : -\partial a \in P(1..i)$ or
(2.3.2) $\exists t \in R_{sd}[q]$ such that
$\forall a \in A(t) : +\partial a \in P(1..i)$ and $t > s$.

Let us work through the condition for $+\partial$, an analogous explanation holds for $-\partial$ below. To show that q is provable defeasibly we have two choices: (1) We show that q is already definitely provable; or (2) we need to argue using the defeasible part of D as well. In particular, we require that there must be a strict or defeasible rule with head q ($r \in R_{sd}[q]$) which can be applied (2.1), i.e., that all the premises in the antecedent of the rule are already provable. But now we need to consider possible "attacks", that is, reasoning chains in support of a complementary of q. To be more

specific: to prove q defeasibly we must show that every complementary literal is not definitely provable (2.2). Also (2.3) we must consider the set of all rules which are not known to be inapplicable and which have head in $\mathcal{I}(q)$ (note that here we consider defeaters, too, whereas they could not be used to support the conclusion q; this is in line with the motivation of defeaters given before). Essentially each such rule s attacks the conclusion q. For q to be provable, each such rule s must be counterattacked by a rule t with head q with the following properties: (i) t must be applicable at this point, and (ii) t must be stronger than s. Thus each attack on the conclusion q must be counterattacked by a stronger rule.

$-\partial$: If $P(i+1) = -\partial q$ then
(1) $-\Delta q \in P(1..i)$ and
(2) (2.1) $\forall r \in R_{sd}[q]\ \exists a \in A(r) : -\partial a \in P(1..i)$ or
(2.2) $\exists p \in \mathcal{I}(q)$ such that $+\Delta p \in P(1..i)$ or
(2.3) $\exists s \in R[\mathcal{I}(q)]$ such that
(2.3.1) $\forall a \in A(s) : +\partial a \in P(1..i)$ and
(2.3.2) $\forall t \in R_{sd}[q]$ either
$\exists a \in A(t) : -\partial a \in P(1..i)$ or $t \not> s$

The purpose of the $-\partial$ inference rules is to establish that it is not possible to prove $+\partial$. This rule is defined in such a way that all the possibilities for proving $+\partial q$ (for example) are explored and shown to fail before $-\partial q$ can be concluded. Thus conclusions tagged with $-\partial$ are the outcome of a constructive proof that the corresponding positive conclusion cannot be obtained.

As we have already said defeaters can be simulated in term of the other elements of Defeasible Logic[5], thus we can consider theories without defeaters. The same is true for the superiority relation, it is possible to give a linear and modular transformation that "compiles" the superiority relation in the rest of the theory, however, the resulting theory is not as natural as with explicit priorities.

3.1.3. *Defeasible Logic at Work*

To explain the mechanism of defeasible derivations we consider the fragment of the contract representing the price policy:

$r_1 : \mathit{AdvertisedPrice}(X, Y), \mathit{Discount}(X, Z), \mathit{Surcharge}(X, W), K = Y - Z + W \Rightarrow \mathit{Price}(X, K)$

$r_2 : \Rightarrow \mathit{Discount}(X, 0)$

$r_3 : \Rightarrow \mathit{Surcharge}(X, 0)$

$r_4 : \mathit{Promotion} \Rightarrow \mathit{Discount}(X, 0)$

$r_5 : \mathit{PremiumCustomer}, \mathit{AdvertisedPrice}(X, Y), Z = Y * .05 \Rightarrow \mathit{Discount}(X, Z)$

$r_6 : \mathit{SpecialOrder}, \mathit{AdvertisedPrice}(X, Y), Z = Y * .05 \Rightarrow \mathit{Surcharge}(X, Z)$

$r_7 : \mathit{PremiumCustomer} \Rightarrow \mathit{Surcharge}(X, 0)$

The first rule states that the price of a good is determined by its advertised price plus the surcharge minus the discount. Rules r_2 and r_3 set the value of discount and surcharge to 0. However those value depend on the status of the goods and customer. Rule r_4 specifies that goods in promotion are not subject to discount, while rule r_5 gives a 5% discount to premium customer. The idea of rules r_6 and r_7 is that special orders are subject to a 5% surcharge unless they are placed by premium customers.

According to the above discussion the superiority relation is as follows:

$$r_2 < r_5 < r_4 \qquad\qquad r_3 < r_6 < r_7$$

and the (p-)incompatile literals are defined such that for every good X the price, discount and surcharge are unique.

Let us examine the following cases: 1) there is a premium customer who has bought a good, and 2) a premium customer placed a special order on a good in promotion. Notice that there is always an applicable instance of rule r_1. Obviously the parameters in it depend on the conditions determined by the other rules. In the first case rules r_2 and r_5 (with the right instances) are both applicable and their conclusion conflict with each other, thus we have to use the superiority relation to resolve the conflict, and we have that r_5 prevails over r_2. For *Surcharge* we have to consider rules r_3, r_6 and r_7, where only r_3 and r_7 are applicable and agree on the second argument of the predicate. Thus we obtain $\mathit{Discount}(X, 5\%)$, and $\mathit{Surcharge}(X, 0)$.

In the second case, as before rule r_5 is applicable, but here we have that rule r_4 is applicable as well. Since r_4 is superior to r_4 we derive that the discount is not applicable (i.e., $\mathit{Discount}(X, 0)$). Since the order is a special order, rule r_6 is applicable, but this rule is defeated by the stronger rule r_7. Thus there is no surcharge ($\mathit{Surcharge}(X, 0)$).

It is worth noting that in this example the superiority relation is explicitly given. In some cases the superiority relation can depend on the context. To accommodate this phenomenon a variant of Defeasible Logic where the superiority relation is computed dynamically according to some principles encoded as defeasible rules has been proposed[2]. We will not pursue this issue any further in this work.

3.2. *Defeasible Deontic Logic*

The version of Defeasible logic we have presented in the previous section does not support explicit reasoning on deontic concepts and is unable to identify the behaviour of roles in the contract and contract violations. On the contrary, monitoring contract performance obviously requires dealing with these aspects. Rather than adding ad hoc predicates to the language, improvements must be made by adding deontic modalities[29,15] and a logic for violations[14], so as to achieve a richer language that can represent the behaviour of roles in the contract in a more natural and applicable manner. The advantage of this approach is to incorporate general and flexible reasoning mechanisms within the inferential engine.

A formal representation language should offer concepts close to the notions the language is designed to capture. Contracts typically contain provisions about deontic concepts such as obligations, permissions, entitlements, violations and other (mutual) normative positions the signatories of a contract agree to comply with. Accordingly a (business) contract language should cater for those notions.

In addition the language should be supplemented by either a formal semantics or facilities to reason with and about the symbols of the language to give meaning to them. As usual the symbols of the language can be partitioned in two classes: logical symbols and extra logical symbols. The logical symbols are meant to represent general concepts and structures common to every contract, while extra logical symbols encode the specific subject matter of given contracts. In this perspective the notions of obligation and permission will be represented by deontic modalities, while concepts such as price, service and so on are better captured by predicates since their meaning varies from contract to contract.

We believe that the approach with deontic modalities is superior to the use of ad hoc predicates at least for the following aspects[b]:

- *Ease of expression and comprehension.* In the modal approach the relationships among the modalities (and normative positions) are encoded in the logic and reasoning mechanism, while for ad hoc predicates contracts are cluttered with rules describing the logical relationships among different modes/representations of one and the same concept. For example, given the predicate $pay(X)$, we have to create predicates such as $obligatory\_pay(X)$, $permitted\_pay(X)$, ... and rules such as $obligatory\_pay(X) \rightarrow permitted\_pay(X)$ and so on. Thus ad hoc predicates do not allow users to focus only and exclusively on aspects related to the content of a contract, without having to deal with any aspects related to its implementation.
- *Clear and intuitive semantics.* It is possible to give a precise, unambiguous, intuitive and general semantics to the notions involved, while each ad hoc predicate requires its own individual interpretation, and in some cases complex constructions (for example reification) are needed to interpret some ad hoc predicates.
- *Modularity.* A current line of research proposes that the combination of deontic modalities with modalities for speech acts and actions faithfully represent complex normative positions such as delegation, empowerment as well as many others that may appear in contracts[24,11]. In the modal approach those aspects can be added or decomposed modularly without forcing the user to rewrite the predicates and rules to accommodate the new facilities, or to reason at different granularity.

[b]In addition to the aspects we discuss here, we would like to point out that it has been agued[20,23] that deontic logic is better than a predicate based representation of obligations and permissions when the possibility of norm violation is kept open. As we argue in the next section a logic of violation is essential for the representation of contracts where rules about violations are frequent.

Thus, we make use of the deontic modalities of obligation and permission. Over the years many approaches to normative reasoning have been proposed and consequently many deontic logics, sometimes with different and incompatible intuitions and motivations, have been developed. See, among others, Hilpinen[22] for a recent overview of the most important approaches. Given the plethora of systems proposed we believe that it is not crucial to provide here a full characterisation of these concepts. However, the majority of the logics assumes at least a logic for obligation enjoys $OA \rightarrow \neg O\neg A$, is closed under logical equivalence and contains the usual schema $OA \equiv \neg P\neg A$ (or equivalently $PA \equiv \neg O\neg A$), where O and P are, respectively, the deontic operators for obligation and permission. Accordingly we will assume a logic that satisfies these minimal principles; other properties can then be added when needed. The formalism is enriched to deal with directed deontic operators: the expression $O_{s,b}A$ states that A is obligatory such that s is the subject of such an obligation and b is its beneficiary[21].

To extend defeasible logic with deontic operators we have two options: the first is to use the same inferential mechanism as basic defeasible logic and to represent explicitly the deontic operator in the conclusion of normative rules[29] while for the second option we introduce new types of rules for the deontic operator to differentiate between normative and definitional (or factual) rules[15].

For example Clause 4.1 "The Purchaser shall follow the Supplier price lists." can be represented as

$$AdvertisedPrice(X) \Rightarrow O_{purchaser,supplier}Pay(X)$$

if we follow the first option and

$$AdvertisedPrice(X) \Rightarrow_{O_{purchaser,supplier}} Pay(X)$$

according to the second option, where $\Rightarrow_{O_{purchaser,supplier}}$ denotes a new type of defeasible rule relative to the deontic operator $O_{purchaser,supplier}$.

The differences between the two approaches, besides the fact that in the first approach there is only one type of rules while the second accounts for factual and deontic rules, is that the first approach has to supplement the definition of p-incompatible literals with appropriate literals for each literal p. The second approach can use different proof conditions for deontic modalities to offer a more fine grained control over the deontic operators and it allows for interaction between deontic operators and other modal operators[15].

3.3. *Logic of Violation*

Finally, let us sketch how to incorporate a logic for dealing with normative violations within the framework we have described so far. A violation of an obligation does not imply the cancellation of such an obligation. This often makes it difficult to characterise the idea of violation in many formalisms for defeasible reasoning[34].

We will take and adapt some intuitions we developed fully elsewhere[14]. To reason aboupt violations we have to represent contrary-to-duties (CTDs) or reparational

obligations. These are in force only when normative violations occur and are meant to "repair" violations of primary obligations[9]. Thus a contrary-to-duty is a conditional obligation arising in response to a violation, where a violation is signalled by an unfulfilled obligation. Very often contracts make provisions about unfulfilled clauses: those provisions describe what some of the subjects of a contract have to do in case they breach the contract (or part of it), and can vary from (pecuniary) penalties –where the obligation to pay the penalty is a contrary-to-duty– to the termination of the contract itself –in this case contrary-to-duties do not exist, i.e., a breach of the contract that cannot be repaired.

3.3.1. *Obligations, Violations and Contrary-to-Duties*

Contrary-to-duties are one of the most debated fields of deontic logic, and, at the same time, they provide a very fertile area for the development of the so called contrary-to-duty paradoxes. In response to the paradoxes many systems often with different intuitions and motivations have been proposed. The question whether an ultimate solution for all CTD paradoxes exists is still open. In this paper we do not touch upon this issue and we focus on a simple logic of violation that seems to avoid most of the well-known paradoxes, offers a simple computational model to compute chains of violations and reparations and can be combined with the Defeasible Deontic Logic of Section 3.2. As we will see the ability do deal with violations or potential violations and the reparational obligation generated from them is one the essential requirements for reasoning about and monitoring the implementation and performance of business contracts.

The idea behind the logic of violation[14] we are going to outline here is that the meaning of a clause of a contract (or, in general a norm in a normative system) cannot be taken in isolation: it depends on the context where the clause is embedded in (the contract). For example a violation cannot exist without an obligation to be violated. The second aspect we have to consider is that a contract is a finite set of explicitly given clauses and, very often, some other clauses are implicit (or can be derived) from the already given clauses. The ability to extract all the implicit clauses from a contract is of paramount importance for the monitoring of it; otherwise some aspects of the contract could be missing from its implementation. Accordingly a logic of violation to be useful for the monitoring and analysis of a contract should provide facilities to i) related interdependent clauses of it and ii) extract or generate all the clauses (implicit or explicit) of a contract.

As we have just discussed a violation cannot exist without an obligation to be violated. Thus we have a sequential order among an obligation, its violation and eventually an obligation generated in response to the violation and so on. To capture this intuition 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"; in other words the interpretation of $OA \otimes OB$, is that A is obligatory, but if the obligation OA is not fulfilled (i.e., when $\neg A$ is the case, and consequently we have a violation

of the obligation OA), then the obligation OB is in force. Elsewhere[14] we discuss that the above interpretation shows that violations are special kinds of exceptions. Several authors have used exceptions to raise conditions to repair a violation in the context of contract monitoring[27,19].

3.3.2. *Reasoning about Violations*

The connective $\otimes$ permits combining 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 (i.e., $A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C$), duplication and contraction on the right, i.e., $A \otimes B \otimes A \equiv A \otimes B$. The right-hand side part of the equivalence states that B is the reparation of the violation of the obligation A. That is, B is in force when $\neg A$ is the case. For the left-had side we have that, as before, a violation of A, i.e., $\neg A$, generates a reparational obligation B, and then the violation of B can be repaired by A. Hoverer, this is not possible since we already have $\neg A$.

One of the features of the logic of violation is to take two rules, or clauses in a contract, and merge them into a new clause. Let examine some common patterns for this kind of construction (the general rule for merging clauses is given in (1) in Section 3.3.3 where we present the logical machinery for it).

Let us consider a policy like (in what follows Γ and Δ are sets of premises, and s and b are respectively the subject and beneficiery of the obligation)

$$\Gamma \Rightarrow O_{s,b}A.$$

Given an obligation like this, if we have that

$$\Delta, \neg A \Rightarrow O_{s',b'}C,$$

then the latter must be a good candidate as reparational obligation of the former. This idea is formalised is as follows:

$$\frac{\Gamma \Rightarrow O_{s,b}A \qquad \Delta, \neg A \Rightarrow O_{s',b'}C}{\Gamma, \Delta \Rightarrow O_{s,b}A \otimes O_{s',b'}C}$$

According to this view, if there exists a conditional obligation whose antecedent is the negation of the propositional content of a different norm, then the latter is a reparational obligation of the former. In this way, the CTD obligation can be forced to be an *explicit reparational obligation* with respect to the violation of its primary counterpart. Accordingly, it seems reasonable to discard both premises when they are subsumed by the conclusion. Their reciprocal interplay makes them two related norms so that they cannot be viewed anymore as independent obligations. Notice that the subjects and beneficiaries of the primary obligation and its reparation can be different, even if very often in contracts they are the same.

Suppose the theory includes (in the following examples the subscripts p and s of the deontic operators refer to the *purchaser* and the *supplier*)

$$r : \mathit{Invoice} \Rightarrow O_{p,s}\mathit{PayWithin7Days}$$

and

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

From these we obtain

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

The schema in (1) can also generate chains of CTDs in order to deal iteratively with violations of reparational obligations. The following case is just an example of this process.

$$\frac{\Gamma \Rightarrow O_{s,b}A \otimes O_{s,b}B \qquad \neg A, \neg B \Rightarrow O_{s,b}C}{\Gamma \Rightarrow O_{s,b}A \otimes O_{s,b}B \otimes O_{s,b}C}$$

For example we can consider the situation described by Clause 5.1 of the contract. Given

$$r : Invoice \Rightarrow O_{s,p} QualityOfService \otimes O_{s,p} Replace3days$$

and

$$r' : \neg QualityOfService, \neg Replace3days \Rightarrow O_{s,p} Refund\&Penalty$$

we derive the new rule

$$r'' : Invoice \Rightarrow O_{s,p} QualityOfService \otimes O_{s,p} Replace3days \otimes O_{s,p} Refund\&Penalty.$$

Given the structure of the inference mechanism it is possible to combine rules in slightly different ways, and in some cases the meaning of the rules resulting from such operations is already covered by other rules in the contract. In other cases the rules resulting from the merging operation are generalisations of the rules used to produce them, consequently, the original rules are no longer needed in the contract. Thus some clauses can be removed from the contract without changing the meaning of it. To deal with this issue we introduce the notion of subsumption between rules. Intuitively a rule subsumes a second rule when the behaviour of the second rule is implied by the first rule. The formal definition of subsumption appropriate for this scenario will be given in Section 3.3.3. Here we illustrate this notion with the help of some examples.

Let us consider the rules

$$r : Invoice \Rightarrow O_{s,p} QualityOfService \otimes O_{s,p} Replace3days \otimes O_{s,p} Refund\&Penalty,$$
$$r' : Invoice \Rightarrow O_{s,p} QualityOfService \otimes O_{s,p} Replace3days.$$

The first rule, r, subsumes the second r'. Both rules state that after the seller has sent an invoice she has the obligation to provide goods according to the published standards, and if she fails to do so (i.e., if she violates such an obligation), then the violation of *QualityOfService* can be repaired by replacing the faulty goods within three days ($O_{s,p} Replace3days$). In other words $O_{s,p} Replace3days$ is a secondary obligation arising from the violation of the primary obligation $O_{s,p} QualityOfService$. In addition r prescribes that the violation of the secondary

obligation $O_{s,p}Replace3days$ can be repaired by $O_{s,p}Refund\&Penalty$, i.e., the seller has to refund the purchaser and in addition she has to pay a penalty.

As we discussed in the previous paragraphs the conditions of a contract cannot be taken in isolation in so far as they exist in a contract. Consequently the whole contract determines the meaning of each single clause in it. In agreement with this holistic view of norms we have that the normative content of r' is included in that of r. Accordingly r' does not add any new piece of information to the contract, it is redundant and can be dispensed from the explicit formulation of the contract.

Another common case of subsumption is exemplified by the rules:

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

It is immediate to recognise that here we have a simple instance of a CTD. The first rule says that after the seller sends the invoice the purchaser has one week to pay it, otherwise the purchaser has to pay the principal plus the interest. Thus we have the primary obligation $O_{p,s}PayWithin7Days$, whose violation is repaired by the secondary obligation $O_{p,s}PayWithInterest$, while, according to the second rule, given the same set of circumstances *Invoice* and $\neg PayWithin7Days$ we have the primary obligation $O_{p,s}PayWithInterest$. However, the primary obligation of r' obtains when we have a violation of the primary obligation of r. Thus the condition of applicability of the second rule includes that of the first rule. Therefore the first rule is more general than the second and we can discard r' from the contract.

We are now ready to describe how the apply the logical machinery we have developed to deal with business contracts.

(1) Given a formal representation of the explicit clauses of a contract we apply the merging mechanism of the logic of violation to generate all implicit conditions that can be derived from the contract.
(2) At this stage we can clean the resulting representation of the contract by throwing away all redundant rules according to the notion of subsumption.
(3) Then we are ready to use defeasible logic to reason about and implement the contract. In particular we can
   (a) detect conflicting rules and solve the resulting conflicts using the superiority relation and
   (b) identify violations and the obligations generated from them, using derivation in defeasible deontic logic.

In the next section we give a formal presentation of the logic of violation and we show how to combine it with Defeasible Deontic Logic. The reader not interested in the mathematical details can skip to Section 4 where we introduce *RuleML* and we show how ot use it to represent contracts in a language that is suitable for machine processing and human reading.

3.3.3. *Defeasible Deontic Logic with Violations*

The first thing to do is to extend the language of Defeasible Deontic Logic to accommodate the new connective $\otimes$. Well-formed-formulas are then defined using the unary connective $\neg$ (negation) and the binary connective $\otimes$ which is intended to formalise CTD statements. However, given the intended interpretation of $\otimes$ we impose some restrictions about its use in rules. Formulas containing $\otimes$ are only permitted in the head of defeasible rules. Thus a rule consists of an antecedent (a set of literals) and a conclusion that can be either a literal of a formula whose main operator is $\otimes$, if the rule is a defeasible rule. Given this the usual rules of contraction, duplication and exchange hold trivially for the antecedent of a rule. However, they do not make any sense for the consequent so that we need properties describing the structural behaviour of $\neg$ and $\otimes$.

The only property we assume for $\neg$ is that it is an involutive operator, i.e., $\neg\neg A \equiv A$ for any formula A; while the basic logical properties for $\otimes$ are the following:

(1) $A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C$
(2) $\bigotimes_{i=1}^{n} A_i \equiv (\bigotimes_{i=1}^{k-1} A_i) \otimes (\bigotimes_{i=k+1}^{n} A_i)$ where $A_j = A_k$ and $j < k$

Condition 1 is just associativity of $\otimes$, while condition 2 corresponds to duplication and contraction. In fact, according to the intuitive reading of this connective given in the previous section, the expression on the right side of $\Rightarrow$ can be considered as an ordered set.

It is possible to give inference rules both for the introduction and elimination of $\otimes$[14]. However, for the purposes of this paper, it is sufficient to define the following inference rule for introducing $\otimes$:

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

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 the two premises into a unique regulation.

As we alluded to in the previous section we use (1) to generate new rules. As soon as we apply it as much as possible we have to drop all redundant rules. This can be done by means of the notion of subsumption given below.

Definition 3.1. Let $r_1 = \Gamma \Rightarrow A \otimes B \otimes C$ and $r_2 = \Delta \Rightarrow D$ be two rules, where $A = \bigotimes_{i=1}^{m} A_i$, $B = \bigotimes_{i=1}^{n} B_i$ and $C = \bigotimes_{i=1}^{p} C_i$. 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 $D = A \otimes \bigotimes_{i=0}^{k \leq p} C_i$.

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 examples above, we can drop rule r, whose normative content is included in r''.

To accommodate the new connective in Defeasible Deontic Logic we have to revise the proof conditions. However, the change needed will affect only the conditions for $+\partial$ and $-\partial$. The proof conditions for definite derivations, $\pm\Delta$, are left unchanged since $\otimes$ is not allowed in the head of strict rules. 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

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

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 $O_{s,b}C$. To capture the potential multiplicity of conclusion of rule we adopt the following notation for rules. 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, $1 \leq i \leq n$, $c_i = q$, and similarly for $R[c_i \in \mathcal{I}(q)]$. Thus for example $r \in R[c_1 = O_{s,b}B]$ 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 any formula incompatible with A; formally $\overline{O_{s,b}A} = \mathcal{I}(A)$. In the simplest case $\overline{O_{s_b}(A)}$ contains just $\neg A$.

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

The above condition is very similar to that presented in Section 3.1.2. 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 $\neg B$ ar 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.

We do not give here the proof condition for $-\partial$. However, we point out that the required modifications mimic the changes done for $+\partial$.

4. Rule Markup Language

RuleML is an XML based language for the representation of rules. It offers facilities to specify different types of rules from derivation rules to transformation rules to reaction rules. Moreover it is capable of specifying queries and inferences in Web ontologies, mappings between Web ontologies, and dynamic Web behaviours of workflows, services, and agents[8].

The *RuleML* initiative[32], started during the Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000) in August 2000, brought together experts from several countries to create an open, vendor neutral XML/RDF-based rule language[8].

The main goal of the initiative is to develop *RuleML* as the canonical web language for rules, based on XML markup, formal semantics and efficient implementations. Its purpose is to allow exchange of rules between major commercial and non-commercial rules systems on the Web and various client-server systems located within large corporations. The *RuleML* initiative involves many organisations to propose *RuleML* as a standard language for exchange of rules to facilitate business-to-customer (B2C) and business-to-business (B2B) interactions over the Web.

From what we have said it is clear that *RuleML* is a generic extensible and semantically neutral rule markup language mainly aimed at the exchange of rules. Accordingly *RuleML* programs are not intended to be executed directly, but the business logic of *RuleML* programs can be implemented via XSLT transformations into the target language of the recipient rule-based systems and then executed.

4.1. *Why* RuleML

RuleML provides a way of expressing business rules in modular stand-alone units. It allows the deployment, execution, and exchange of rules between different systems and tools. It is expected that *RuleML* will be the declarative method to describe rules on the Web and distributed systems[36].

RuleML arranges rule types in an hierarchical structure comprising reaction rules (event-condition-action-effect rules), transformation rules (functional-equational rules), derivation rules (implicational-inference rules), facts ('premiseless' derivation rules, i.e., derivation rules with empty bodies), queries ('conclusionless' derivation rules, i.e., derivation rules with empty heads) and integrity constraints (consistency-maintenance rules). Each part of a rule is an expression that has specific functions in the rule.

The *RuleML* Hierarchy first directly branches out into two categories: Reaction Rules and Transformation Rules. Transformation Rules then break down into

Derivation Rules, that, in turn, subdivide into Facts and Queries. Finally, Queries break down into Integrity Constraints[32]. Refer to Figure 1 for the *RuleML* hierarchy.

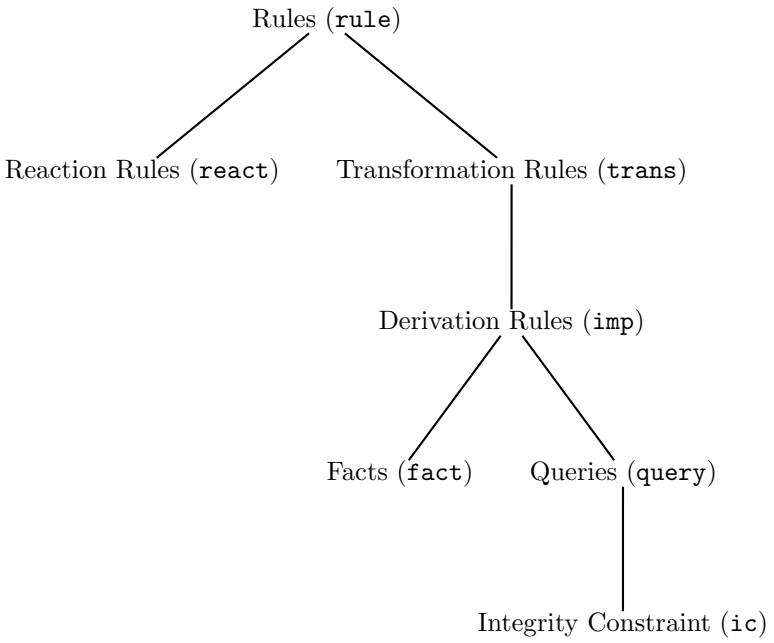

Fig. 1. *RuleML* Hierarchy

However in this paper we will only focus on derivation rules and facts since, here we are interested in a conceptual representation of contracts. In general the distinction among the types of rules is more pragmatic than conceptual and is geared towards the actual implementation of the rules themselves, and very often several types of rules can be transformed into other classes of rules straightforwardly[8,32].

Conflicts among rules are very common in contracts; thus facilities to deal with them are essential in any language designed to represent contracts. *RuleML* offers two ways to prioritise rules (and then solve conflicts): quantitative priorities and qualitative priorities. A quantitative priority is a numerical *Priority* property for a rule; it states the salience of a rule. On the other hand a qualitative priority is a binary relation defined over the set of rule labels and determines the relative strength of two rules[36]. An alternative approach is to supplement the syntax of *RuleML* with an empty `<superiority>` element with two attributes whose values are the names of the rules the element refers to[3].

However, as we said, *RuleML* is not without limitations. It does not support the use of modality and it is unable to deal with violations. As such, improvements must be made to cover these aspects if one wants to use it to represent business contracts.

4.2. *Fundamentals of* RuleML

In this section we are going to explore the building blocks of *RuleML* and we are going to identify the additions required to faithfully represent contracts.

4.2.1. *Premises and Conclusions*

The first thing we have to consider is the representation of predicates (atoms) to be used in premises or conclusions in *RuleML*. A predicate is an n-ary relation and it is defined as an `<Atom>` element in *RuleML* with the following DTD definition[c]

```
<!ELEMENT Atom  (Not?,Rel,(Ind|Var)*)>
<!ELEMENT Not   >
<!ELEMENT Rel   (#PCDATA)>
<!ELEMENT Var   (#PCDATA)>
<!ELEMENT Ind   (#PCDATA)>
```

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 proposition the element represents is true). *RuleML* contains two types of negation, classical negation and negation as failure[35,8]. However, negation as failure can be simulated by other means in Defeasible Logic[6], so we do not include it in our syntax.
Accordingly

```
<Atom>
      <Rel>DeliverWithinOneDay</Rel>
      <Ind>Good</Ind>
      <Ind>PurchaseOrderDate</Ind>
      <Ind>DeliveryDate</Ind>
</Atom>
```

is a predicate that is true when the supplier has delivered the `Good` specified by a purchaser in a purchase order with date `PurchaseOrderDate` the day after the reception of the purchase order, stored in the value of the ground instance of `DeliveryDate`.

4.2.2. *Reaction Rules*

Reaction Rules represent a key member of the *RuleML* family of languages. They deal with invoking actions that are triggered as a reaction to an event. They define the way the system reacts to changes in the environment and communication by specifying the *event*, *pre-* and *post-* conditions and the actions that can be triggered

[c]Although the current version of *RuleML* (Version 0.87) is based on XML schema, here, due to space limitations, we will give the XML grammar using simplified DTD definitions.

by the event. A reaction rule can only be applied in a forward directional manner in a natural fashion, i.e., it first checks for the events and conditions and then executes the action where the events and conditions have been met. Thus they can also be called Event-Condition-Action-(Effect) rules[8,36,32]. Currently reaction rules have not been specified in *RuleML*.

4.2.3. *Transformation Rules*

Transformation rules were introduced in *RuleML* 0.81. They generally reuse the same concrete syntax as other rule types, specifying the condition in the "`<body>`". Transformation rules consist of a transformation invoker, a condition, and a transformation return.

Transformation rules have the following syntax:

```
<!ELEMENT Trans  (head,body,foot)>
<!ELEMENT head   (Atom)>
<!ELEMENT body   (And)>
<!ELEMENT foot   (Atom)>
<!ELEMENT And    (Atom)+>
```

The `<Trans>` element is the top-level element denoting a transformation rule. It uses the transformation invoker role "`head`", followed by an optional condition role "`body`", and lastly the transformation return role "`foot`", meaning if condition specified in the `<body>` holds then conclusion in the `<head>` will be transformed to the value in the `<foot>` section[8]. We are not going to consider transformation rules in the present paper for the representation of contracts. However, a transformation rule can be represented by two derivation rules (see the next subsection) where the first derivation rule has the same `<body>` and `<head>` as the transformation rule while the `<body>` and `<head>` of the second derivation rule are, respectively, the `<head>` and the `<foot>` of the transformation rule.

4.2.4. *Derivation Rules*

Derivation Rules are special transformation rules that like characteristic functions on success just return true[32]. An alternative way to understand derivation rules to consider them as special reaction rules whose action is to add a *conclusion* when certain *conditions* have been met. They comprise one or more conditions but derive only one conclusion. These rules can be applied in a forward or backward manner, the latter reducing the proof of a goal (conclusion) to proofs of all its subgoals (conditions)[8].

Derivation Rules allow the derivation of information from existing rules[35]. They are able to capture concepts not stored explicitly from 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.

Derivation rules have the following syntax:

```
<!ELEMENT Imp    ((head,body)|(body|head))>
<!ELEMENT body   (And)>
<!ELEMENT head   (Atom)>
<!ELEMENT And    (Atom)+>
```

A derivation rule has two roles, *Condition* (`<body>`) and *Conclusion* (`<head>`); the latter being an atomic predicate logic formula, and the former a conjunction of formulas[36], meaning that derivation rules consist of one more conditions and a conclusion. Accordingly the above example can be represented as follows:

```
<Imp label="PremiumCustomerRule">
   <body>
      <And>
         <Atom>
               <Rel>TotalExpense</Rel>
               <Var>Customer</Var>
               <Var>Expense<Var>
         </Atom>
         <Atom>
               <Rel>Greater</Rel>
               <Var>Expense</Var>
               <Ind>$10000</Ind>
         </Atom>
      </And>
   </body>
   <head>
      <Atom>
            <Rel>PremiumCustomer</Rel>
            <Var>Customer</Var>
      </Atom>
   </head>
</Imp>
```

4.2.5. *Facts*

Facts are considered as special derivation rules but without the specification of conjunction of *premises* or *conditions* "`body`"[8]. They denote simple pieces of information that are deemed to be true. URLs/URIs can also be embedded within facts to reference the elements that are being referred to.

Facts have the following syntax:

```
<!ELEMENT Fact (Atom)>
```

A `<Fact>` element uses just a conclusion role "`head`", meaning whatever that is included in the "`head`", is understood as true[8]. Clauses 3.1 and 3.2 of the contract can be deemed facts, thus, for example, the representation of Clause 3.1 is:

```
<Fact label="2.1">
   <Atom>
        <Rel>CommencementDate</Rel>
        <Ind>2002-01-30</Ind>
```

```
   </Atom>
</Fact>
```

4.2.6. *Queries*

Queries are like derivation rules with the conclusion (`<head>`) empty[8]. Queries can either be proved backward as top-down goals or forward via 'goal-directed' bottom-up processing[32].

Queries have the following syntax:

```
<!ELEMENT Query (body)>
```

A `<Query>` element uses just the conditions role "`body`", meaning it is a conclusionless derivation rule[8], and can be used to obtain the set of tuples satisfying the condition represented by the body.

4.2.7. *Integrity Constraints*

Integrity constraints also known as integrity rules are special reaction rules that alert the system whenever an inconsistency has been detected after the event/condition has been fulfilled. There are two types of Integrity Constraints: State Constraints and Process Constraints. A state constraint must hold at all times. A process constraint is related to the dynamic integrity of the system. It interrupts the process of a triggered action transiting from one state of the system process to another[8,36,35].

Integrity constraints have the following syntax:

```
<!ELEMENT Ic (body)>
```

An element `<Ic>` represents an integrity constraint. The conditions specified in the `<body>` are used to check if any inconsistency has happened without the need to recognise any event. They were classified as a special case of reaction rules different from derivation rules[36]. In the current version of *RuleML* integrity conditions are just denials (derivation rules with empty head). Thus integrity constraints are meant to represent situations that must be prevented in a system. Defeasible logic can use its own reasoning mechanism to avoid inconsistency.

Grosof[17] suggests the introduction of a `<mutex>` element (for "mutually exclusive") to represent the notion of p-incompatible literals. However, mutex literals are just a special case of integrity constraints. Accordingly there is no need to introduce a new tag for them, and consequently the specifications of p-incompatible literals can be represented by `<Ic>` elements. For example p-incompatible literals for the predicate (`<rel>`) `Price` can be defined in the following way:

```
<Ic label='price'>
   <And>
      <Atom>
         <Rel>Price</Rel>
         <Var>Good</Var>
         <Var>X</Var>
```

```
      </Atom>
      <Atom>
         <Rel>Price</Rel>
         <Var>Good</Var>
         <Var>Y</Var>
      </Atom>
      <Atom>
         <not>
            <eq>
               <Var>X</Var>
               <Var>Y</Var>
            </eq>
         </not>
      </Atom>
   </And>
</Ic>
```

As we have already discussed the price for a good is unique, thus the above integrity constraint indicates a non legal state when we have that one and the same goods (`Good`) is related via the the predicate `Price` to two different values `X` and `Y`.

5. *RuleML* Derivation Rules and Defeasible Logic

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. In this paper we will concentrate only on derivation rules and integrity constraints. For the relationships between *RuleML* and Defeasible Logic we will translate derivation rules (`Imp`s) into defeasible rules in defeasible logic and we will limit ourselves to integrity constraints (`Ic`s) corresponding to mutex that will be translated to p-incompatible literals specifications. In this perspective a derivation rule

```
<Imp label='label'>
  <body>
    ...
  </body>
  <head>
    ...
  </head>
</Imp>
```

is transformed into a defeasible rule

$$\texttt{label} : \texttt{body} \Rightarrow \texttt{head}.$$

A possible limitation of this approach is that all rules are defeasible. There are no strict rules. In contracts we have two types of clauses: definitional and operational clauses. Definitional clauses define the meaning of the terms used in the contract while operational clauses describe the behaviour of the contract. Typically definitions are not defeasible while the behaviour is defeasible. Let us discuss some possible solutions for this shortcoming.

The first solution is to use both strict and defeasible rules: this can be achieved by using `<Imp>` for strict rules and we can introduce a new type of rule `<Def>` for defeasible rules[3], or we can dispense with strict rules by observing that in a proper definition, i.e., a definition that does not admit exceptions or all exceptions have been catered for, sufficient conditions are monotonic. This also means that it is not possible to have rules whose conclusion is incompatible with the definiendum. We notice that for these cases defeasible rules behave as strict rules.

It has been argued that rule languages are not the most appropriate mean to represent definitions of terms. Thus rule languages and reasoning must be supplemented and integrated with ontologies and logic suitable for reasoning about ontologies. Grosof and Poon[19] propose the integration of *RuleML* and ontologies to deal with business contracts. However, there is no discussion how to integrate ontologies and rules. Antoniou[1] suggests the idea of using a Description Logic oracle to derive strict terms and a Defeasible logic reasoner for non-monotonic derivations. If we allow definitions to be defeasible (which is often the case in normative reasoning) then the situation is more complicated and we have to account for interactions between the monotonic and non-monotonic parts of a contract. For example Wang at al.[10,37] put forth combinations of non-monotonic formalisms and description logic where the extension of strict literals (i.e., definitions) can be updated by new instances derived in a non-monotonic fashion, while Governatori[12] introduces a seamless integration of (weakly expressive) defeasible logic and description logic. We recognise here the importance of those aspects but we will not purse these issues any further in this paper. We leave them for future extensions.

6. Contracts in *RuleML*

Any representation language should offer concepts closely related to those present in the phenomenon the representation language is intended to capture. As we have already noted, contracts contain normative concepts such as obligations, permissions, violations. It has been argued that normative reasoning, by its own nature, is defeasible. 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`.

```
<!ELEMENT And         (Atom|Obligation|Permission)*>
<!ELEMENT Head        (Atom|Obligation|Permission)+>
<!ELEMENT Obligation  (Not?,Rel,(Ind|Var)*)>
<!ATTLIST Obligation  subject IDREFS beneficiary IDREFS>
<!ELEMENT Permission  (Not?,Rel,(Ind|Var)*)>
<!ATTLIST Permission  subject IDREFS beneficiary IDREFS>
```

The above grammar allows us to introduce obligations and permissions, and, in combination with priorities, gives us the ability to represent contracts more faithfully. For example, Clause 5.2 can be represented by the following rule:

```
<Imp label="5.2">
   <body>
      <And>
         <Atom>
               <Rel>PurchaseOrder</Rel>
               <Var>Good</Var>
               <Var>PurchaseOrderDate</Var>
         </Atom>
     </And>
   </body>
   <head>
      <Obligation subject="Supplier" beneficiary="Purchaser">
            <Rel>DeliverWithinOneDay</Rel>
            <Var>Good</Var>
            <Var>PurchaseOrderDate</Var>
            <Var>DeliverDate</Var>
       </Obligation>
   </head>
</Imp>
```

In a similar way Clause 4.1 is represented by the rule

```
<Imp label="4.1" href="http://supplier.com/catalog.htm">
   <body>
      <And>
         <Atom>
                <Rel>PurchaseOrder</Rel>
                <Ind>Purchaser</Ind>
                <Ind>Supplier</Ind>
                <Var>Good</Var>
         </Atom>
         <Atom>
                <Rel>AdvertisedPrice</Rel>
                <Ind>Supplier</Ind>
                <Var>Good</Var>
                <Var>Price</Var>
         </Atom>
     </And>
   </body>
   <head>
      <Obligation subject="Purchser">
             <Rel>PurchaseOrderPrice</Rel>
             <Var>Good</Var>
             <Var>Price</Var>
      </Obligation>
   </head>
</Imp>
```

This rule states that if there is a purchase order issued by the purchaser about a particular good, and the price of good is advertised in the supplier catalogue published at the URI in the `href` attribute of the rule, then the price of the good in the purchase order should be the advertised price.
To illustrate how to encode the superiority relation we consider the second part of Clause 6.1

```
<Imp label="6.1b">
<body>
  <And>
    <Atom>
      <Rel>PurchaseOrder</Rel>
      <Var>Good</Var>
      <Var>Price</Var>
      <Var>Date</Var>
    </Atom>
  </And>
</body>
<head>
  <Obligation subject="Purchaser" beneficiary="Supplier">
      <Rel>Pay</Rel>
      <Var>Good</VAr>
      <Var>Price</Price>
  </Obligation>
</head>
</Imp>
<Imp label="6.1c">
<body>
  <And>
    <Atom>
      <Rel>WrittenAgreement</Rel>
      <Var>Good</Var>
      <Var>Price</Var>
    </Atom>
 </And>
</body>
<head>
  <Obligation subject="Purchaser" beneficiary="Supplier">
    <Rel>Pay</Rel>
    <Var>Good</Var>
    <Var>Price</Price>
 </Obligation>
</head>
</Imp>
```

The above two rules can conflict with each other when the price stated in them for one and the same good is different. To resolve this conflict we have to assess the relative strength of the two rules. This is achieved by a fact where the head states that rule 6.1c overrides rule 6.1b:

```
<Fact>
   <head>
      <Atom>
            <Rel>Override</Rel>
            <Ind href="6.1c"/>
            <Ind href="6.1b"/>
      </Atom>
   </head>
</Fact>
```

However, so far, we cannot deal with violations and the obligations arising in re-

sponse to them. This type of construction occurs very frequently in contracts, and it is very important for the correct (automatic) execution and monitoring of e-contracts. To this end we propose to replace the content of the `<head>` element of normative rules with 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 $\otimes$. Also in this case we refine the notion of head.

```
<!ELEMENT head      (Atom|Obligation|Permission|Behaviour)>
<!ELEMENT Behaviour ((Obligation)+,Permission?)>
```

As an illustration of this construction consider the first part of Clause 6.1 where the reparation to the violation is stated in the same clause as the main obligation:

```
<Imp label="6.1a">
   <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 * 1.07</Var>
         </Obligation>
      </Behaviour>
   </head>
</Imp>
```

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) –just look at the formulation of Clause 5.3. Thus it can be convenient to have facilities to represent violations directly.

```
<!ELEMENT And       (Atom|Obligation|Permission|Violation)*>
<!ELEMENT Violation  >
<!ATTLIST Violation  rule IDREF>
```

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[19,27]

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.

Clause 5.3 contains a disjunction in the body. Thus we can split it into two rules where we can use the violations of Clause 4.1 and Clause 4.2 in the respective bodies. Here we show one of the two resulting rules; the other has the same structure and the only difference is the value of the attribute of the `<Violation>` element.

```
<Imp label="5.3a">
   <body>
      <And>
        <Violation rule="4.1"/>
      </And>
   </body>
   <head>
      <Permission subject="Purchaser" beneficiary="Supplier">
         <Rel>Charge100DollarsPerHour</Rel>
      </Permission>
   </head>
</Imp>
```

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.

```
<!ELEMENT Behaviour ((Obligation+,Reparation)|(Obligation*,Permission?))>
<!ELEMENT Reparation >
<!ATTLIST Reparation penalty IDREF>
<!ELEMENT Penalty   ((Obligation+,Reparation)|(Obligation*,Permission?))>
```

If this strategy is chosen then we can rewrite Clause 5.2 and Clause 5.3 as follows

```
<Imp label="5.2alternative">
   <Body>
      <And>
         <Atom>
                 <Rel>PurchaseOrder</Rel>
                 <Var>Good</Var>
                 <Var>PurchaseOrderDate</Var>
         </Atom>
     </And>
   </Body>
   <Head>
      <Obligation subject="Supplier" beneficiary="Purchaser">
            <Rel>DeliverWithinOneDay</Rel>
            <Ind>Supplier</Ind>
            <Ind>Purchser</Ind>
```

```
            <Var>Good</Var>
            <Var>PurchaseOrderDate</Var>
            <Var>DeliveryDate</Var>
       </Obligation>
       <Reparation penalty="5.3"/>
   </Head>
</Imp>

<Penalty label="5.3">
      <Permission subject="Supplier" beneficiary="Purchaser">
         <Rel>Charge100Hour</Rel>
      </Permission>
</Penalty>
```

6.1. *Deontic* RuleML *and Defeasible Deontic Logic*

The idea of the translation here is the same as in Section 5. The only differences concern the representation of the deontic tag. `<Obligation>` and `<Permission>` are translated to O and P. The `<Behaviour>` element contains a sequence of two or more deontic elements, thus

```
<Imp label="r">
  <body>...</body>
  <head>
    <Behaviour>
      <Obligation>A1</Obligation>
      ...
      <Deontic>An</Deontic>
    </Behaviour>
  </head>
</Imp>
```

corresponds to

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

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

The remaining deontic tags, i.e., `<Reparation>`, `<Penalty>` and `<Violation>`, do not increase the expressive power of the language but are included as convenient shortcuts. `<Reparation>` and `<Penalty>` occur in pairs, where the `penalty` attribute of `<Reparation>` refers to the `label` of the `<Penalty>` element. Hence they can be combined in the rule where the `<Reparation>` element occur using the $\otimes$ operator.

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

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

Finally, 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 Defeasible Deontic Logic, namely

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

Eventually the two rules can be combined via the $\otimes$I rule (1) in

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

7. Reasoning about Contracts in *RuleML*

The first step in processing a contract written in natural language is to provide its logical representation. To this end all the clauses of the contract are transformed into facts, definitions and normative rules. A normative rule is a single rule with a

conjunctive body and head a sequence of obligations and permissions. This representation can be given in a language that is suited to computers as well as humans. At this stage the contract is ready to be processed in order to derive all conditions included in it and eventually to detect inconsistencies and loopholes. Very often contracts contain conditions that are implicitly stated in the clauses of the contract but that can be derived from the explicit clauses. Thus at this stage we apply the introduction rule ($\otimes$I) to normalise the contract until we reach a fixed-point (i.e., when no further new rules can be derived). The result of normalisation can produce redundant rules in the sense that the content/behaviour of a rule is part of the content/behaviour of a more specific rule; hence the first rule is no longer required to implement the contract, and can be safely removed. In this step we "throw away" all rules subsumed (according to Definition 3.1) by some other rules in the contract. After the subsumption step the contract can be fed in a *RuleML* engine to execute or monitor the contract performance at run time.

8. Conclusions

Business Contracts are used to specify the modalities that the signatories should be held responsible to, and to state the actions and penalties to be undertaken in the event of a violation. We showed how to transform the contract from its implicit to its explicit form to enable precise Contract Monitoring via the use of computers. In particular, we formed *RuleML* to be an appropriate language in the context of Contract Monitoring. Different choices than *RuleML* are available, such as *BCL* (Business Contract Language)[27] and *XrML*[25]. All these are XML dialects and suitable candidates for Contract Monitoring tasks. However, conflicts in contract rules may occur. *RuleML* supports the usage of priorities amongst rules for conflict resolution and so it serves as a better choice as compared to the other options, which are deficient in it.

RuleML has been extended to deal explicitly with deontic concepts and violations. Through the use of Deontic logic, modalities, roles and behaviours of the contract have been defined. Defeasible Logic on the other hand has helped clarify inconsistencies and derived additional information that has not been specifically defined in the contract. It has also provided solutions to the resolution of conflicts comprised within the contract rules. We illustrated how different rules can be merged together using the $\otimes$ connective to form a single simplified rule. This process has helped in the elimination of redundancy and enhanced the efficiency in the coding of the language that the contract will be implemented in. From the logical analysis in Deontic and Defeasible Logic, contract rules have been implemented within the body and head of the *RuleML* structure. In this way, the logical representation of the contract can be implemented into a machine executable syntax using *RuleML*.

Acknowledgements

A preliminary version of this paper was presented at Jurix 2004[16].

I would like to thank Antonino Rotolo, Zoran Milosevic and Ee Wen Phang for the many fruitful discussions we had on the topic of this paper and related issues, and Leon van der Torre and Mehedi Dastani for their insights on the logic of violation and its potential applications.

Thanks are also due to the anonymous referees for their valuable criticisms and comments on early versions of this paper.

This work was partially supported by Australia Research Council under Discovery Project No. DP0558854 on "A Formal Approach to Resource Allocation in Web Service Oriented Composition in Open Marketplaces".

References

1. Grigoris Antoniou. Nonmonotonic rule system on top of ontology layer. In I. Horrocks and J. Hendler, editors, *ISWC 2002*, number 2432 in LNCS, pages 394–398. Springer-Verlag, Berlin, 2002.
2. Grigoris Antoniou. Defeasible logic with dynamic priorities. *International Journal of Intelligent Systems*, 19(5):463–472, 2004.
3. Grigoris Antoniou, Antonis Bikakis, and Gerd Wagner. A system for nonmonotonic rules on the Web. In Grigoris Antoniou and Harold Boley, editors, *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004*, number 3323 in LNCS, pages 23–36. Springer-Verlag, Berlin, 2004.
4. Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. A flexible framework for defeasible logics. In *Proc. American National Conference on Artificial Intelligence (AAAI-2000)*, pages 401–405. AAAI/MIT Press, Menlo Park, CA, 2000.
5. Grigoris Antoniou, David Billington, Guido Governatori, and Michael J. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2(2):255–287, 2001.
6. Grigoris Antoniou, Micheal J. Maher, and David Billington. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming*, 41(1):45–57, 2000.
7. Nick Bassiliades, Grigoris Antoniou, and Ioannis Vlahavas. DR-DEVICE: A defeasible logic system for the Semantic Web. In Hans Jürgen Ohlbach and Sebastian Schaffert, editors, *2nd Workshop on Principles and Practice of Semantic Web Reasoning*, number 3208 in LNCS, pages 134–148. Springer-Verlag, Berlin, 2004.
8. Harold Boley, Said Tabet, and Gerd Wagner. Design rationale for RuleML: A markup language for Semantic Web rules. In Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, editors, *Proceedings of SWWS'01, The first Semantic Web Working Symposium*, pages 381–401, 2001.
9. José Carmo and Andrew J.I. Jones. Deontic logic and contrary to duties. In D.M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic. 2nd Edition*, volume 8, pages 265–343. Kluwer, Dordrecht, 2002.
10. Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded semantics for description logic programs in the Semantic Web. In Grigoris Antoniou and Harold Boley, editors, *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004*, number 3323 in LNCS,

pages 81–97. Springer-Verlag, Berlin, 2004.
11. Jonathan Gelati, Guido Governatori, Antonino Rotolo, and Giovanni Sartor. Normative autonomy and normative co-ordination: Declarative power, representation, and mandate. *Artificial Intelligence and Law*, In print, 2005.
12. Guido Governatori. Defeasible description logic. In Grigoris Antoniou and Harold Boley, editors, *Rules and Rule Markup Languages for the Semantic Web: Third International Workshop, RuleML 2004*, number 3323 in LNCS, pages 98–112. Springer-Verlag, Berlin, 2004.
13. Guido Governatori, Michael J. Maher, David Billington, and Grigoris Antoniou. Argumentation semantics for defeasible logics. *Journal of Logic and Computation*, 14(5):675–702, 2004.
14. Guido Governatori and Antonino Rotolo. A Gentzen system for reasoning with contrary-to-duty obligations. a preliminary study. In Andrew J.I. Jones and John Horty, editors, Δ*eon'02*, pages 97–116, London, May 2002. Imperial College. `http://eprint.uq.edu.au/archive/00001893/01/deon02.pdf`
15. Guido Governatori and Antonino Rotolo. Defeasible logic: Agency, intention and obligation. In Alessio Lomuscio and Donald Nute, editors, *Deontic Logic in Computer Science*, number 3065 in LNAI, pages 114–128, Berlin, 2004. Springer-Verlag, Berlin 2004.
16. Guido Governatori and Antonino Rotolo. Representing contracts using RuleML. In Thomas Gordon, editor, *Legal Knowledge and Information Systems*, volume 120 of *Frontieres in Artificial Intelligence and Applications*, pages 141–150. IOS Press, Amsterdam, 2004.
17. Benjamin N. Grosof. Representing e-commerce rules via situated courteous logic programs in RuleML. *Electronic Commerce Research and Applications*, 3(1):2–20, 2004.
18. Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In *Proceedings of the 1st ACM Conference on Electronic Commerce (EC99)*, pages 68–77. ACM Press, 1999.
19. Benjamin N. Grosof and Terrence C. Poon. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the 12th International Conference on World Wide Web*, pages 340–349. ACM Press, 2003.
20. Henning Herrestad. Norms and formalization. In *ICAIL'91: Proceedings of the 3rd International Conference on Artificial Intelligence and Law*, pages 175–184. ACM Press, 1991.
21. Henning Herrestad and Christen Krogh. Obligations directed from bearers to counterparts. In *ICAIL'95: Proceedings of the 5th International Conference on Artificial Intelligence and Law*, pages 210–218. ACM Press, 1995.
22. Risto Hilpinen. Deontic logic. In Lou Goble, editor, *The Blackwell Guide to Philosophical Logic*, number 4 in Blackwell Philosophy Guides, chapter 8, pages 159–182. Blackwell, Oxford, 2001.
23. Andrew J. I. Jones and Marek Sergot. On the characterization of law and computer systems: the normative systems perspective. In John-Jules Ch. Meyer and Roel J. Wieringa, editors, *Deontic logic in computer science: normative system specification*, pages 275–307. John Wiley and Sons Ltd., 1993.
24. Andrew J. I. Jones and Marek Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
25. Jae Kyu Lee and Mye M. Sohn. The eXtensible Rule Markup Language. *Communications of the ACM*, 46(5):59–64, May 2003.
26. Michael J. Maher, Andrew Rock, Grigoris Antoniou, David Billignton, and Tristan

Miller. Efficient defeasible reasoning systems. *International Journal of Artificial Intelligence Tools*, 10(4):483–501, 2001.

27. Zoran Milosevic, Simon Gibson, Peter F. Linington, James Cole, and Sachin Kulkarni. On design and implementation of a contract monitoring facility. In B. Benatallah, editor, *First IEEE International Workshop on on Electronic Contracts*, pages 62–70. IEEE Press, 2004.
28. Lee Naish. A declarative debugging scheme. *Journal of Functional and Logic Programming*, 3, 1997.
29. Donald Nute. Norms, priorities and defeasibility. In Paul McNamara and Henry Prakken, editors, *Norms, Logics and Information Systems. New Studies in Deontic Logic*, pages 83–100. IOS Press, Amsterdam, 1998.
30. Jeremy Pitt, Lloyd Kamara, and Alexander Artikis. Interaction patterns and observable commitments in a multi-agent trading scenario. In *AGENTS'01: Proceedings of the 5th International Conference on Autonomous Agents*, pages 481–488. ACM Press, 2001.
31. Henry Prakken and Giovanni Sartor. A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law*, 4:331–368, 1996.
32. RuleML. The Rule Markup Initiative, 26th February 2005. `http://www.ruleml.org`
33. Guglielmo Simari and Ron Loui. A mathematical treatment of argumentation and its implementation. *Artificial Intelligence*, 53:125–157, 1992.
34. Leon van der Torre and Yao-Hua Tan. The many faces of defeasibility. In Donald Nute, editor, *Defeasible Deontic Logic*, pages 79–121. Kluwer, Dordrecht, 1997.
35. Gerd Wagner. 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, June 2002.
36. Gerd Wagner, Said Tabet, and Harold Boley. MOF-RuleML: The abstract syntax of RuleML as a MOF model. In *OMG Meeting*, Boston, October 2003.
37. Kewen Wang, David Billington, Jeff Blee, and Grigoris Antoniou. Combining description logic and defeasible logic for the semantic web. In Grigoris Antoniou and Harlod Boley, editors, *RuleML 2004*, volume 3323 of *LNCS*, pages 170–181. Springer-Verlag, Berlin, 2004.