

A Semantic Web Based Architecture for e-Contracts in Defeasible Logic

Guido Governatori and Duy Pham Hoang

School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, QLD 4072
email: {guido,pham}@itee.uq.edu.au

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

1 Introduction

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

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

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

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

One of the main benefits of this approach is that we can use formal methods to reason with and about the clauses of a contract. In particular we can

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

Secondly, a language for contracts should be conceptual. This, following the well-known *Conceptualization Principle* of [13], effectively means that the language should allow their users to focus only and exclusively on aspects related to the content of a contract, without having to deal with any aspects related to their implementation.

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

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

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

For example the following fragment of a contract of service taken from [8] are definitional clauses

3.1 A “Premium Customer” is a customer who has spent more than \$10000 in goods.

3.2 Service marked as “special order” are subject to a 5% surcharge. Premium customers are exempt from special order surcharge.

while

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

and

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

are normative clause. The above contract clauses make it clear that there is the need to differentiate over Clauses 3.1 and 3.2 on one side, and Clauses 5.2 and 5.3 on the other. The first two clauses are factual/definitional clauses describing states of affairs, defining notions in the conceptual space of the contract. For example clause 3.1 defines the meaning of “Premium Customer” for the contract,

and Clause 3.2 gives a recipe to compute the price of services. On the other hand Clauses 5.2 and 5.3 state the (expected) legal behaviour of the parties involved in the transaction. In addition there is a difference between Clause 5.2 and Clause 5.3. Clause 5.2 determines an obligation for one of the party; on the other hand Clause 5.3 establishes a permission. Hence, according to our previous discussion about the functionalities of the representation formalism, a logic meant to capture the semantics of contracts has to account for such issues. Since the seminal work by Lee [16] Deontic Logic has been regarded as one of the most prominent paradigms to formalise contracts. In [8] we further motivate the need of deontic logic to capture the semantics of contracts and the reasons to choose it over other formalisms.

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

From a logical perspective every clause of a contract can be understood as a rule where we have the conditions of applicability of the clause and the expected behaviour. Thus we have that we can represent a contract by a set of rules, and, as we have already argued, these rules are non-monotonic. Thus we need a formalism that is able to reason within this kind of scenario. Our choice here is Defeasible Logic (we will motivate this choice in section 2).

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

The paper is organised as follows: in Section 2 we present the logic on which the DR-CONTRACT architecture is based. Then in Section 3 we explain the extension of *RuleML* corresponding to the logic of the previous section, and we establish a mapping between the two languages. Then, in Section 4 we discuss the system architecture of the DR-CONTRACT framework. Finally we relate our work to similar approaches and we give some insights about future developments in Section 5.

2 Defeasible Deontic Logic of Violation

For a proper representation of contracts and to be able to reason with and about them we have to combine and integrate logics for various essential components of contracts. In particular we will use the Defeasible Deontic Logic of Violation (DDL_V) proposed in [8]. This logic combines deontic notions with defeasibility and violations. More precisely DDL_V is obtained from the combination of three logical components: Defeasible Logic, deontic concepts, and a fragment of a logic

to deal with normative violations. Before presenting the logic we will discuss the reasons why such notions are necessary for the representation of contracts.

In [14] Courteous Logic Programming (CLP) has been advanced as the inferential engine for business contracts represented in *RuleML*. Here, instead, we propose Defeasible Logic (DL) as the inferential mechanism for *RuleML*. In fact, CLP is just a notational variant of one of the many logics in the family proposed by [3,1] (see [4] for the relationships between DL and CLP). Accordingly, it may be possible to integrate the extensions we develop in the rest of the paper within a CLP framework. Over the years DL proved to be a flexible non-monotonic formalism able to capture different and sometimes incompatible facets of non-monotonic reasoning [1], and efficient and powerful implementations have been proposed [3,18,6,5]. The primary use of DL in the present context is aimed at the resolution of conflicts that might arise from the clauses of a contract; in addition DL encompasses other existing formalisms proposed in the AI & Law field (see, [9]), and recent work shows that DL is suitable for extensions with modal and deontic operators [10,12].

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

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

$$Price(SpamFilter, 50).$$

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

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

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

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

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

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

$$\begin{aligned} r : PremiumCustomer(X) &\Rightarrow Discount(X) \\ r' : SpecialOrder(X) &\Rightarrow \neg Discount(X) \end{aligned}$$

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 strict¹ or defeasible rule for p .

For a full presentation of Defeasible Logic see [2,8],

The next step is to integrate deontic logic in defeasible logic. To this end we follow the idea presented in [10]. In the context of contract we introduced the directed deontic operators $O_{s,b}$ and $P_{s,b}$. Thus, for example the expression $O_{s,b}A$ means that A is obligatory such that s is the subject of such an obligation and b is its beneficiary; similarly for $P_{s,b}$, where $P_{s,b}A$ means that s is permitted to do A in the interest of b . In this way it is possible to express rules like the following

$$PurchaseOrder \Rightarrow O_{Supplier, Purchaser} Deliver_Within1Day$$

that encodes Clause 5.2 of the contract presented above.

Finally, let us sketch how to incorporate a logic for dealing with normative violations within the framework we have described so far. A violation occurs when an obligation is disattended, thus $\neg A$ is a violation of the obligation OA . However, a violation of an obligation does not imply the cancellation of such an obligation. This makes often difficult to characterise the idea of violation in many formalisms for defeasible reasoning [22]. We will take and adapt some intuitions we developed fully in [11]. To reason on violations we have to represent contrary-to-duties (CTDs) or reparational obligations. As is well-known, these last are in force only when normative violations occur and are meant to “repair” violations of primary obligations. In the spirit of [11] we introduce the non-classical connective \otimes , whose interpretation is such that $OA \otimes OB$ is read as “ OB is the reparation of the violation of OA ”. The connective \otimes permits to combine primary and CTD obligations into unique regulations. The operator \otimes is such that $\neg\neg A \equiv A$ for any formula A and enjoys the properties of associativity,

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

duplication and contraction. For the purposes of this paper, it is sufficient to define the following rule for introducing \otimes :²

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

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

Suppose the theory includes

$$\begin{aligned} r : \text{Invoice} &\Rightarrow O_{s,b}\text{Pay_Within7Days} \\ r' : \neg\text{Pay_Within7Days} &\Rightarrow O_{s,b}\text{Pay_WithInterest}. \end{aligned}$$

From these rules we obtain

$$r'' : \text{Invoice} \Rightarrow O_{s,b}\text{Pay_Within7Days} \otimes O_{s,b}\text{Pay_WithInterest}.$$

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

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

1. $\Gamma = \Delta$ and $D = A$; or
2. $\Gamma \cup \{\neg A_1, \dots, \neg A_m\} = \Delta$ and $D = B$; or
3. $\Gamma \cup \{\neg B_1, \dots, \neg B_n\} = \Delta$ 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 example above, we can drop rule r , whose normative content is included in r'' .

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

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

Provability is based on the concept of a *derivation*. A derivation is a finite sequence $P = (P(1), \dots, P(n))$ of tagged literals satisfying four conditions (which correspond to inference rules for each of the four kinds of conclusion). Here we

² The \otimes is allowed only in the head of defeasible rules. See [8] for a full motivation of this design choice.

will give only the conditions for $+\Delta$ and $+\partial q$. $P(1..i)$ denotes the initial part of the sequence P of length i :

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

To accommodate the new connective (\otimes) in DDLV we have to revise the inference mechanism of Defeasible Logic. The first thing we have to note is that now a defeasible rule can be used to derive different conclusions. For example given the rule

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

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

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

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

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

The above condition is very similar to the same condition for basic defeasible logic [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 $\neg B$ or any other formula incompatible with B (Clause 2.1.2). A similar explanation holds true for Clause 2.3.2 where we want to show that a rule does not support an attack on the intended conclusion.

3 Contracts in RuleML

In order to integrate the DR-CONTRACT engine with Semantic Web technology we decided to use RuleML [20] as an open and vendor neutral XML/RDF syntax

for contracts. We tried to re-use as many features of standard RuleML syntax as possible. However, since some notions essential for the representation of contracts are not present in standard RuleML we have created our DR-CONTRACT DTD (Figure 1).³

```

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

```

Fig. 1. DR-CONTRACT Basic DTD

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

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

³ Although the current version of RuleML (Version 0.89) is based on XML Schema, here due to space limitation and for ease of presentation, we will give the XML grammar using simplified DTD definitions.

true). RuleML contains two types of negation, classical negation and negation as failure [23,7]. However, negation as failure can be simulated by other means in Defeasible Logic [4], so we do not include it in our syntax.

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

Derivation rules allow the derivation of information from existing rules [23]. They are able to capture concepts not stored explicitly in the existing information. For example, a customer is labelled as a “Premium” customer when he buys \$10000 worth of goods. As such, the rule here states that the customer must have spent \$10000 on goods, thus deriving the information here that the customer is a “Premium” customer. A derivation rule has an attribute **strength** whose value ranges over **strict** and **defeasible** and it denotes the type of rule to be associated to it when computed in defeasible logic.

A derivation rule has two immediate sub-elements, *Condition* (`<Body>`) and *Conclusion* (`<Head>`); the latter being either an atomic predicate formula or a sequence of obligations, and the former a conjunction of formulas [24], meaning that derivation rules consist of one more conditions and a conclusion.

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

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

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

is transformed into a defeasible rule

$$r : \text{body} \Rightarrow OA_1 \otimes \dots \otimes \mathbf{X}A_n$$

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

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

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

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

The new deontic tags in the DR-CONTRACT extended DTD in Figure 2 –<Reparation>, <Penalty> and <Violation>– do not increase the expressive

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

Fig. 2. DR-CONTRACT Extended DTD

power of the language but are included as convenient shortcuts. It is possible to express a violation explicitly by saying that a particular rule is triggered in response to a violation (i.e., when an obligation is not fulfilled) . However, it can be convenient to have facilities to represent violations directly –just look at the

formulation of Clause 5.3. In general a violation can be one of the conditions that trigger the application of a rule. Accordingly a **<Violation>** element can be included in the body of a rule. A violation cannot subsist without a rule that is violated by it. Hence the attribute **rule** is a reference to the rule that has been violated. Many contract languages [15,19] contain similar constructions. The activation of such constructions/processes requires the generation of a violation event/literal. On the contrary our approach does not require it. All we have to do is to check for a sequence of literals joined with the \otimes operator where the initial part of the sequence is not satisfied.

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

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

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

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

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

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

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

For example given the following fragment of a contract

```

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

```

```

<Penalty label="p">
  <Obligation>B1</Obligation>
  ...
  <Deontic>Bm</Deontic>
</Penalty>

```

the rule corresponding to it is

$$r : \text{body} \Rightarrow OA_1 \otimes \dots \otimes OA_n \otimes OB_1 \otimes \dots \otimes XB_m.$$

4 DR-CONTRACT System Architecture

The system architecture of DR-CONTRACT is inspired by the system architecture of the family of DR-DEVICE applications [21,6,5] and consists of four main modules (see Figure 3):

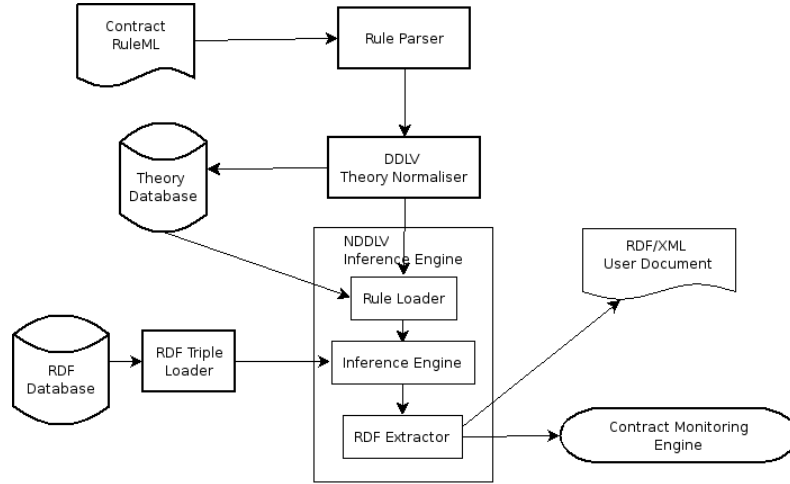


Fig. 3. DR-CONTRACT System Architecture

1. A Rule Parser to transform a DR-CONTRACT compliant document (a contract) into a theory to be passed to the next module. The parser is based on the XML processor and it is rather similar in nature to the Logic Loader module of the DR-Device family applications [21,6,5].

2. A DDLV normaliser. The normaliser takes as input a DDLV theory (obtained from the previous step) and iteratively merges rules in the theory according to the inference rule 1 and then removes rules subsumed by a more general rule according to Definition 1. It repeats the cycle till it reaches the fixed-point of such a construction (which is guaranteed to exist and to be unique [11]). Once a theory has been normalised the normal form is saved to a repository (for faster loading in successive calls), and the normalised theory NDDLTV is passed to the DDLV engine. In addition the normaliser applies a transformation that removes superiority relation by compiling it into new rules (the technique used here is similar to that of [2]).
3. The RDF loader downloads/queries the input documents, including their schemata, and it translates the RDF descriptions into fact objects according to the RDF-NDDLTV translation schema based on the DR-CONTRACT DTD.
4. The NDDLTV inference engine consists of two components:
 - The *Rule Loader* compiles the rules in a NDDLTV theory in objects. We distinguish two types of objects: Rules and Literals or atoms. Each rule object has associated to it a list of (pointers to) modal literals (corresponding to head of the rule) and a set of (pointers to) modal literals implemented as an hash table. Each atom object has associated to it four hash tables: the first with pointers to the rules where the atom occurs positively in the head, the second with pointers to the rules where the atom occurs negatively in the head, the third with pointers to the rules where the atom occurs positively in the body and the last with pointers where the atom occurs negatively in the body.
 - The *Inference Engine* is based on an extension of the *Delores* algorithm/implementation proposed in [18] as a computational model of Basic Defeasible Logic. In turn:
 - It asserts each fact (as an atom) as a conclusion and removes the atom from the rules where the atom occurs positively in the body, and it “deactivates” the rules where the atom occurs negatively in the body. The complement of the literal is removed from the head of rules where it does not occur as first element. The atom is then removed from the list of atoms.
 - It scans the list of rules for rules where the body is empty. It takes the first element of the head and searches for rule where the negation of the atom is the first element. If there are no such rules then, the atom is appended to the list of facts, and removed from the rules
 - It repeats the first step.
 - The algorithm terminates when one of the two steps fails. On termination the algorithm outputs the set of conclusions.⁴

⁴ Notice that the algorithm runs in linear time. Each atom/literal in a theory is processed exactly once and every time we have to scan the set of rules, thus the complexity of the above algorithm is $O(|L| * |R|)$, where L is the set of distinct modal literals and R is the set of rules.

5. Finally the conclusions are exported either to the user or to a monitoring contract facility such as BCL [19,17] as an RDF/XML document through an RDF extractor.

5 Conclusion and Related Works

In this paper we have presented a system architecture for a Semantic Web based system for reasoning about contracts. The architecture is inspired by the system architecture of the DR-DEVICE family of applications. The main differences between our approach and the DR-DEVICE is in the use of an extended variant of Defeasible Logic. The extensions are in the use of modal operator and a non classical operator for violations. The same difference applies for the SweetDeal approach by Grosz [14,15]. We have also argued that the extension with modal (deontic) operators is not only conceptually sound but also necessary to capture the semantics of contracts. In the same way the implementation of the inference engine is an extension of the algorithm used by Delores [18] to cope with deontic operators and the \otimes operator.

The handling of temporal aspects is a very delicate matter in contract monitoring. The current architecture does not cover temporal reasoning. However, [12] proposes an extension of Defeasible Logic that can represent and reason with temporalised normative positions. In particular the framework offers facilities to initiate and terminate obligations, permissions, prohibitions and other complex normative positions. We have planned to study how to efficiently incorporate such features in our Deontic Defeasible Logic of Violations.

Currently we have implemented prototypes of the inference engine in Python and Java, and experimental results show that the Python implementation is able to deal with some of the benchmark theories of [18] with theories in some case with over 50000 rules.

Acknowledgements

This work was partially supported by the UQ Early Career Researcher Grant no. 2004001458 on “A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies”.

References

1. G. Antoniou, D. Billington, G. Governatori, and M. Maher. A flexible framework for defeasible logics. In *(AAAI-2000)*, 401–405. AAAI/MIT Press, 2000.
2. G. Antoniou, D. Billington, G. Governatori, and M. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2(2):255–287, 2001.
3. G. Antoniou, D. Billington, G. Governatori, M. Maher, and A. Rock. A family of defeasible reasoning logics and its implementation. In W. Horn, editor, *ECAI 2000*, 459–463. IOS Press, 2000.

4. G. Antoniou, M. Maher, and D. Billington. Defeasible logic versus logic programming without negation as failure. *J. of Logic Programming*, 41(1):45–57, 2000.
5. N. Bassiliades, G. Antoniou, and I. Vlahavas. A defeasible logic reasoner for the semantic web. In G. Antoniou and H. Boley, editors, *RuleML 2004*, LNCS 3323, 49–64. Springer-Verlag, 2004.
6. N. Bassiliades, G. Antoniou, and I. Vlahavas. DR-DEVICE: A defeasible logic system for the Semantic Web. In H.J. Ohlbach and S. Schaffert, editors, *2nd PPSWR*, LNCS 3208, 134–148. Springer-Verlag, 2004.
7. H. Boley, S. Tabet, and G. Wagner. Design rationale for ruleml: A markup language for semantic web rules. In I.F. Cruz, S. Decker, J. Euzenat, and D.L. McGuinness, editors, *SWWS'01*, 381–401, 2001.
8. G. Governatori. Representing business contracts in RuleML. *Int. J. of Cooperative Information Systems*, 14(2-3):181–216, 2005.
9. G. Governatori, M. Maher, D. Billington, and G. Antoniou. Argumentation semantics for defeasible logics. *J. of Logic and Computation*, 14(5):675–702, 2004.
10. G. Governatori and A. Rotolo. Defeasible logic: Agency, intention and obligation. In A. Lomuscio and D. Nute, editors, *Deontic Logic in Computer Science*, LNAI 3065, 114–128. Springer-Verlag, 2004.
11. G. Governatori and A. Rotolo. Logic of Violations: A Gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 2005.
12. G. Governatori, A. Rotolo, and G. Sartor. Temporalised normative positions in defeasible logic. In A. Gardner, editor, *10th ICAIL*, 25–34. ACM Press, 2005.
13. J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036, 1982.
14. B.N. Grosz. Representing e-commerce rules via situated courteous logic programs in RuleML. *Electronic Commerce Research and Applications*, 3(1):2–20, 2004.
15. B.N. Grosz and T. C. Poon. SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *12th WWW*, 340–349. ACM Press, 2003.
16. R.M. Lee. A logic model for electronic contracting. *Decision Support Systems*, 4:27–44, 1988.
17. P. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, and S. Neal. A unified behavioural model and a contract for extended enterprise. *Data & Knowledge Engineering*, 51:5–29, 2004.
18. M. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller. Efficient defeasible reasoning systems. *Int. J. of AI Tools*, 10(4):483–501, 2001.
19. Z. Milosevic, S. Gibson, P.F. Linington, J. Cole, and S. Kulkarni. On design and implementation of a contract monitoring facility. In *1st IWEC*, 62–70. IEEE Press, 2004.
20. RuleML. The Rule Markup Initiative, 1 September 2005.
21. T. Skylogiannis, G. Antoniou, N. Bassiliades, and G. Governatori. DR-NEGOTIATE – a system for automated agent negotiation with defeasible logic-based strategies. In *EEE'05*, 44–49. IEEE Press, 2005.
22. L. van der Torre and Y.-H. Tan. The many faces of defeasibility. In D. Nute, editor, *Defeasible Deontic Logic*, 79–121. Kluwer, 1997.
23. G. Wagner. How to design a general rule markup language. In *Proceedings of XML Technology for the Semantic Web (XSW 2002)*, LNI 14, 19–37. GI, 2002.
24. G. Wagner, S. Tabet, and H. Boley. MOF-RuleML: The abstract syntax of RuleML as a MOF model. In *OMG Meeting*, Boston, 2003.