Guido Governatori and Antonino Rotolo 'Modelling Contracts Using RuleML' in T. Gordon (ed.) *Legal Knowledge and Information Systems. Jurix 2004: The Seventeenth Annual Conference*. Amsterdam IOS Press, 2004, pp. 141-150.

Modelling Contracts Using *RuleML*

Guido Governatori[1] Antonino Rotolo[2]
[1] *School of ITEE, The University of Queensland, Brisbane, Australia*
Email: guido@itee.uq.edu.au
[2] *CIRSFID and Law Faculty, University of Bologna, Bologna, Italy*
E-mail: rotolo@cirfid.unibo.it

Abstract. This paper presents an approach for the specification and implementation of e-contracts for Web monitoring. This is done in the setting of *RuleML*. We argue that monitoring contract execution requires also a logical account of deontic concepts and of violations. Accordingly, *RuleML* is extended to cover these aspects.

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 hold responsible for and to state the actions or penalties that may be taken when any of the stated agreements is 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, so that all relevant tasks of contracts can be described as business processes, where business processes are defined 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 contracts and they are also hidden in many programs across clients, applications and database tiers of modern business information systems, which makes 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 represented implicitly in the form of natural language expressions. This implicit form cannot be applied to information systems; rules must be made operational by transforming them into a declarative (rule) language.

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, in its human-oriented form, will be analysed and represented in a logical form using Deontic and Defeasible Logic. From its logical form, the contract will be transformed into a machine readable rule notation, based on *RuleML*, and implemented as executable semantics.

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 [6]. 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 (see, e.g., [13, 6]). However, we are not interested here in modelling the process of making contracts. Rather, we will focus on 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:

2 Definitions and Interpretations NOT SHOWN TO SAVE SPACE

3 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.

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 the (Goods) are available to the (Purchaser) under Quality of Service Agreement High (http://supplier/qos1.htm).
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). Formally, 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.

3 The Logical Framework

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

RuleML [1] 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 in fact a way of expressing business rules as modular, stand-alone units [15, 16]. It also possesses the ability to resolve conflicts using priorities and override predicates [5, 16].

In [10] 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 one of the many variants of DL [4]. Over the years DL proved to be a flexible nonmonotonic formalism able to capture different and sometimes incompatible facets of nonmonotonic reasoning [2], and efficient and powerful implementations have been proposed [3]. 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, [7]), and recent work shows that DL is suitable for extensions with modal and deontic operators [9]. 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 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, "John is a minor". *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 "every minor is a person": $minor(X) \rightarrow person(X)$. *Defeasible rules* are rules that can be defeated by contrary evidence. An example of such a rule is "every person has the capacity to perform legal acts to the extent that the law does not provide otherwise": $person(X) \Rightarrow hasLegalCapacity(X)$. The idea is that if we know that someone is a person, then we may conclude that she has legal capacity *unless there is other evidence suggesting that she may not have*. 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 $r : person(X) \Rightarrow hasLegalCapacity(X)$ and $r' : minor(X) \Rightarrow \neg hasLegalCapacity(X)$ which contradict one another, no conclusive decision can be made about whether a minor has legal capacity. But if we introduce a superiority relation $>$ with $r' > r$, then we can indeed conclude that the minor does not have legal capacity. 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. Provability is based on the concept of a *derivation*. A derivation is a finite sequence of tagged literals satisfying four conditions (which correspond to inference rules for each of the four kinds of conclusion). Here is only an informal explanation of the conditions required for $+\Delta q$ and $+\partial q$ [2]. For a literal q to be definitely provable we need to find a strict rule with head q, of which all antecedents have been definitely proved previously. 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 also the defeasible part of the theory. In particular, we require that there must be a strict or defeasible rule with head q, which can be applied. But now we need to consider possible "attacks", that is, reasoning chains in support of $\sim q$. To be more specific: to prove q defeasibly we must show that $\sim q$ is not definitely provable. Also, we must consider the set of all rules which are not known to be inapplicable and which have head $\sim q$.

RuleML is not without limitations. It 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 on contract performance obviously requires to deal with these aspects. Rather than adding ad hoc predicates to the language, improvements must be made by adding deontic modalities in DL. This is indeed possible, as shown in [9]. Also, a logic for violations [8] is required, to achieve a richer language that can represent the behaviour of roles in the contract in a more applicable manner. The advantage is to incorporate general and flexible reasoning mechanisms within the inferential engine.

Thus, we make use of the deontic modalities of obligation and permission. It is not crucial to provide here a full characterisation of these concepts. It is sufficient to assume at least that the logic for obligation enjoys $OA \rightarrow \neg O \neg A$, is closed under logical equivalence and contains the usual schema $OA \equiv \neg P \neg A$. 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.

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 makes often difficult to characterise the idea of violation in many formalisms for defeasible reasoning [14]. We will take and adapt some intuitions we developed fully in [8]. 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 [8] we introduce the non-classical connective $\otimes$, whose interpretation is such that $OA \otimes OB$ is read as "OB is the reparation of the violation of OA". The connective $\otimes$ permits to combine primary and CTD obligations into unique regulations. The operator $\otimes$ is such that $\neg\neg A \equiv A$ for any formula A and enjoys the properties of associativity, duplication and contraction. For the purposes of this paper, it is sufficient to define the following rule for introducing $\otimes$ (where $\Rrightarrow$ stands either for $\rightarrow$ or $\Rightarrow$):

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

where $\mathbf{X}$ denotes an obligation or a permission. In this last case, we will impose that D is an atom. Since the minor premise states that $\mathbf{X}_{s,b}D$ is a reparation for $O_{s,b}B_n$, i.e. the last literal in the sequence $\bigotimes_{i=1}^{n} O_{s,b}B_i$, we can attach $\mathbf{X}_{s,b}D$ to such sequence. In other words, this rule permits to combine into a unique regulation the two premises. Suppose the theory includes $r : \mathit{Invoice} \Rightarrow O_{s,b}\mathit{Pay\_Within7Days}$ and $r' : \neg \mathit{Pay\_Within7Days} \Rightarrow O_{s,b}\mathit{Pay\_WithInterest}$. From these we obtain $r'' : \mathit{Invoice} \Rightarrow O_{s,b}\mathit{Pay\_Within7Days} \otimes O_{s,b}\mathit{Pay\_WithInterest}$. 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* $r_1 = \Gamma \Rrightarrow \bigotimes_{i=1}^{m} A_i \otimes B$ *and* $r_2 = \Gamma' \Rrightarrow C$ *be two rules. Then* r_1 subsumes r_2 *iff 1)* $\Gamma = \Gamma'$ *and* $\bigotimes_{i=1}^{m} O_{s,b}A_i = C$*; or 2)* $\Gamma \cup \{\neg A_1, \ldots, \neg A_m\} = \Gamma'$ *and* $B = (C \otimes D)$.

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''.

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 [5]. The *RuleML* initiative [1], 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 [5]. 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 is working with 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.

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 [16]. *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 holds 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 [1]. 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. 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 [16]. 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.

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[1]

```
<!ELEMENT Atom  (Rel,(Ind|Var)*)>
<!ELEMENT Rel   (#PCDATA)>
<!ELEMENT Var   (#PCDATA)>
<!ELEMENT Ind   (#PCDATA)>
```

Accordingly

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

[1]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.

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 (`DeliveryDate`) the reception of the purchase order.

Derivation Rules Derivation Rules are 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) [5]. Derivation Rules allow the derivation of information from existing rules [15]. They are capable to capture concepts not stored explicitly from the existing information. For example, a customer is labelled as a "Premium" customer when he buys $88 worth of coffee in a café restaurant. As such, the rule here states that the customer must spent $88 on coffee, 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 is a conjunction of formulas [16], 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>CustomerCoffeExpense</Rel>
               <Var>Customer</Var>
               <Var>Expense<Var>
         </Atom>
         <Atom>
               <Rel>Greater</Rel>
               <Var>Expense</Var>
               <Ind>$88</Ind>
         </Atom>
      </And>
   </body>
   <head>
      <Atom>
            <Rel>PremiumCustomer</Rel>
            <Var>Customer</Var>
      </Atom>
   </head>
</Imp>
```

Facts Facts are considered as special derivation rules but without the specification of conjunction of *premises* or *conditions* "`body`" [5]. 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 is included in the "head" is understood as true [5]. Clause 3.1 and 3.2 of the contract can be deemed as 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>
```

5 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 contains normative concepts such as obligations, permission, violations, and, it has been argued that, by its own nature, normative reasoning 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 on 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  (Rel,(Ind|Var)*)>
<!ATTLIST Obligation  subject IDREFS beneficiary IDREFS>
<!ELEMENT Permission  (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 more faithfully contracts. 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>
```

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 be in conflict 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 an `Override` fact which states that rule 6.1c overrides rule 6.1b:

```
<Fact>
   <Atom>
      <Rel>Override</Rel>
      <Ind>6.1c</Ind>
      <Ind>6.1b</Ind>
   </Atom>
</Fact>
```

However, so far, we cannot deal with violations and the obligations arising in response to them. This type of construction occurs very frequently in contract, 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. This Clause contains a disjunction in the antecedent, thus it can be split into two rules with the same head:

```
<Imp label="5.3a">
   <body>
      <And><Rel>Violation</Rel>
           <Ind>5.1</Ind>
      </And>
   </body>
   <head>
      <Permission subject="Purchaser" beneficiary="Supplier">
         <Rel>Charge100DollarsPerHour</Rel>
      </Permission>
   </head>
</Imp>
```

6 Reasoning about Contracts in *RuleML*

The first step to process 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 the head is a sequence of obligations and permissions. This representation can be given in a language that is suitable 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 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.

7 Conclusions

In this paper we showed how to transform contracts from their implicit to their explicit form to enable precise Contract Monitoring via the use of computers. We identified *RuleML* to be an appropriate language for this task. Other choices are available, such as *BCL* (Business Contract Language) [12] and *XrML* [11]. All these are XML dialects suitable for Contract Monitoring. However, conflicts in contract rules may occur. *RuleML* supports the usage of rule priorities 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 deontic logic, modalities, roles and behaviours of the contract have been defined. DL 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 using $\otimes$ connective to form single simplified rules. 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, 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*. Contract modelling is crucial in the AI & Law field. At any rate, the aim of this paper was just to focus on semantic web components such as *RuleML*, and to discuss it as a standard for contract monitoring on the web. Future research is also aimed at implementing and testing a prototype based on this framework.

References

[1] RuleML. The Rule Markup Initiative. `www.ruleml.org`. Date 22nd October 2004.

[2] G. Antoniou, D. Billington, G. Governatori and M. Maher. A flexible framework for defeasible logics. In *Proc. AAAI-2000*. AAAI/MIT Press, 2000.

[3] G. Antoniou, D. Billington, G. Governatori, M. Maher and A. Rock. A Family of Defeasible Reasoning Logics and its Implementation. In *ECAI 2000*. IOS Press, 2000.

[4] G. Antoniou, M.J. Maher and D. Billington. Defeasible Logic versus Logic Programming without Negation as Failure, *Journal of Logic Programming*, 42, 2000.

[5] H. Boley, S. Taber, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. *Proc. SWWS'01*, Stanford, July/August 2001.

[6] J. Gelati, G. Governatori, A. Rotolo, and G. Sartor. Normative Autonomy and Normative Coordination: Declarative Power, Representation, and Mandate. Forthcoming in *AI & Law*.

[7] G. Governatori, M. Maher, D. Billington, and G. Antoniou. Argumentation semantics for defeasible logics. *Journal of Logic and Computation*, 14, 2004.

[8] G. Governatori and A. Rotolo. A Gentzen System for Reasoning with Contrary-To-Duty Obligations. A preliminary Study. In *Proc. Deon 2002*. London, 2002.

[9] G. Governatori and A. Rotolo. Defeasible Logic: Agency, Intention and Obligation. In *Proc. Deon'04*. LNAI 3065. Springer, 2004.

[10] B. Grosof. Representing E-Business Rules in the Semantic Web: Situated Courteous Logic Programs in RuleML. *WITS'01*. New Orleans, December, 2001.

[11] J.K. Lee and M.M. Sohn. The eXtensible Rule Markup Language. *Comm. of the ACM*, 46, 2003.

[12] Z. Milosevic, S. Gibson, P.F. Linington, J. Cole, and S. Kulkarni. On Design and Implementation of a Contract Monitoring Facility. In B. Benatallah et al. (eds.), *Proceedings of WEC*. IEEE, 2004.

[13] J. Pitt, L. Kamara, and A. Artikis. Interaction Patterns and Observable Commitments in a Multi-Agent Trading Scenario. In J. Müller et al. (eds.), *Proc. of AA*. ACM, 2001.

[14] L. van der Torre, L. and Y. Tan. The many faces of defeasibility. In *Defeasible Deontic Logic*. Kluwer, 1997.

[15] G. Wagner. How to Design a General Rule Markup Language. Eindhoven University of Technology, Faculty of Technology Management, June 2002.

[16] G. Wagner, S. Tabet, and H. Boley. MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model. *OMG Meeting*, October 2003, Boston.