Towards a Formalisation of Electronic Contracting Environments

Nir Oren
Dept. of Computer Science
King's College London
Strand, London
WC2R 2LS, United Kingdom

Sofia Panagiotidi
Univ. Politecnica de Catalunya
Edifici OMEGA-S206
C/ Jordi Girona 1-3
E - 08034 Barcelona, Spain

Javier Vázquez-Salceda
Univ. Politecnica de Catalunya
Edifici OMEGA-317
C/ Jordi Girona 1-3
E - 08034 Barcelona, Spain

Sanjay Modgil
Dept. of Computer Science
King's College London
Strand, London
WC2R 2LS, United Kingdom

Michael Luck
Dept. of Computer Science
King's College London
Strand, London
WC2R 2LS, United Kingdom

Simon Miles
Dept. of Computer Science
King's College London
Strand, London
WC2R 2LS, United Kingdom

Abstract

The IST-CONTRACT project is in the process of creating an electronic contracting language. One of the goals of this language is that it has formal underpinnings, and formalizations at a number of levels have been created. One of the lowest levels, upon which the other levels are built is the normative level. At this level, we identify how contract clauses (modeled as norms) may evolve over time. In this paper, we describe this formalization, and show how we may associate various states with a norm throughout its lifecycle. We also show how more complex evaluations may be carried out over a norm, and conclude with an example showing the application of the framework over a contract and its associated norms.

Introduction

With the increasing popularity of online transactions, the need for electronic contracting has become apparent. While generic contracts are applicable in many situations, and violations sufficiently rare that a human may resolve disputes, the appearance of web-services, and the need to regulate interactions between them highlights the desirability of fully automated contracting. Such fully automated contracting requires the ability to describe a contract in a machine interpretable way, ideally in a form over which inference may be performed. Additionally, techniques for automatically generating and enforcing contracts are also required, as well as protocols allowing agents to create and modify contracts.

Work dealing with some of these areas already exists; for example, (Jennings et al. 2001) discusses automated negotiation in various contexts, including in contracts, while (Grosof and Poon 2004; Kollingbaum 2005) and (Dignum et al. 2002) all suggest different types of contracting languages. At their core, contracts are primarily normative documents; that is they impose a set of (possibly conditional) requirements on an agent behavior. These requirements may range from actions that the agent may, or should, undertake, to states of affairs within the environment that an agent may, should, or should not, allow to occur. To formalize a contracting language, one must thus first formalize its normative components. As discussed later, researchers have provided many such formalizations, often in the context of deontic logic. Our interest in norms is more focused; as part of a contracting language, we are interested in tracking their changing state (for example, when they are "active", as well as the more traditional "violated"). Furthermore, our application domain requires slightly different philosophical assumptions when compared to those made in the deontic tradition, as we assume that norms can be violated, but may then, in some cases, be "un-violated".

In this paper, we present a formal normative framework that allows us to track the changing status of norms. The remainder of the paper is structured as follows: we begin by providing an informal overview of our normative model. We then describe a typical case in which norms may be used within a contract. This serves as a running example throughout the remainder of the paper. The normative model is then formalized. This is carried out in two parts. We begin by structurally describing the various elements of our model, and then show how we may capture their dynamic behavior. After illustrating our model via an example, we conclude by discussing related and future work.

Norms for modeling Contract Clauses

We assume that a contract is made up of various descriptive elements, for example, stating which ontologies may be used to explain the terms found within it. Most importantly, it specifies a set of *clauses*, each of which represents a *norm*.

Norms can be interpreted as socially derived prescriptions specifying that some set of agents (the norm's *targets*) may, or must, perform some action, or see that some state of affairs occurs. Norms can be understood as regulating the behavior of agents. This is their role when encoded in contracts.

Norms are social constructs, and we believe that it is meaningless to consider norms independently of their social aspect. This is because a norm is imposed on the target by

some other entity (the imposer) which must be granted, via the society, some power to impose the norm. Without this power, the norm's target is free to ignore the norm's prescriptions. With the presence of power, a penalty may be imposed on an agent violating a norm. These penalties take on the form of additional norms, giving certain agents within a society *permission* to impose penalties (or *obliging* them to do so).

When designing our normative model, we attempted to meet the following requirements, imposed upon us by the contracting domain in which we operate:

- The model should allow for the monitoring of norms. That is, it should allow for the determination of whether a violation took place and, if possible, who was responsible for causing the violation.
- Verification of norms should also be supported, i.e. determining whether conflicts between norms could occur, or whether a norm could never / sometimes / always be complied with.
- Agents should be able to make use of the normative model to support their own practical reasoning, i.e. deciding which action they should undertake.
- Norms should be able to cope with contrary to duty obligations as well as conditions based on the status of other norms. For example, consider the pair of norms "One is obliged to park legally", and "If one parks illegally, one is obliged to pay a fine". The second norm carries normative weight only if the first norm is violated. These types of norms commonly appear within contracts, and it is thus critical that our model is able to represent them.
- The model should be extensible, allowing different knowledge representations and reasoning mechanisms to make use of it.

No requirement was placed on detecting and resolving normative conflict. Many such techniques exist, each having a different view of what constitutes normative conflict (e.g., (Vasconcelos, Kollingbaum, and Norman 2007)). It is intended that these techniques make use of our framework for their underlying representation of norms. Similarly, our model should not prescribe what must occur when a violation is detected. Instead, we assume that the contract would contain clauses dealing with such situations.

Since norms may have normative force only in certain situations, we associate norms with an *activation condition*. Norms are thus normally *abstract*, and are *instantiated* when the norm's activation condition holds. Once a norm has been instantiated, it remains active, irrespective of its activation condition, until a specific *expiration condition* holds. When it occurs, the norm is assumed to no longer have normative force. Finally, independent of these two conditions is the norm's *normative goal*, which is used to identify when the norm is violated (in the case of an obligation), or what the agent is actually allowed to do (in the case of a permission). Obligations and permissions are the two *norm types* on which our framework focuses. Like others, we assume that additional norm types may be constructed from these basic types (e.g. a prohibition could be seen as an obligation with a negated normative goal).

Norms may be activated, met and discharged based on a number of factors including the status of other norms, the state of the environment (and the actions performed by other agents therein), and the status of contracts.

Example: Car Insurance Brokerage

As a running example, we make use of a simplified version of a use case scenario from the IST-CONTRACT project. This scenario models the agreements between several parties in the car insurance domain. The goal of this use case is to enhance the quality and efficiency of the total damage claims handling process between parties.

After the repair company receives a damaged car and consents to the repair, two parties, a *car insurance broker* (*Damage Secure*) and a *repair company*, officially commit to a short term contract which specifies the details of the repairing procedure, including the invoice etc. Then, the repair company repairs the car and notifies Damage Secure when it is complete. Damage Secure handles the payment agreed to in the contract provided there is no dispute over the quality of the repair (in which case an expert is called to perform a quality assessment).

The focus of interest of this example is to show how a repair contract and a set of instantiated norms over the repair of a car function within the the domain and normative environment environment. Such a case is useful to demonstrate how a contract and the norms attached to it can be monitored throughout its execution.

Formalization

In this section, we formalize our notions of norms. We do so in a number of steps: first, we define their structure; after this is done, we show how the status of a norm may change over time. Before examining norms, we must define a number of related concepts.

Formal Preliminaries

We assume the use of a predicate based first order language $\mathcal{L}$ containing logical symbols: connectives $\{\neg, \wedge, \vee, \rightarrow\}$, quantifiers $\{\forall, \exists\}$, an infinite set of variables, and the non-logical predicate, constant and function symbols. The standard definitions for free and bound variables, as well as ground formulas are assumed. Finally, the set of well formed formulas of $\mathcal{L}$ are denoted as $\mathit{wff}(\mathcal{L})$. A single well-formed formula from this set is denoted *wff*.

We make use of the standard notions of substitution of variables in a wff, where $S = \langle t_1/v_1, \ldots, t_n/v_n \rangle$ is a substitution of the terms $t_1, \ldots, t_n$ for variables $v_1, \ldots, v_n$ in a wff. If no variables exist in a wff resulting from a substitution, it is said to be fully grounded, and is partially grounded otherwise.

Our model allows us to infer predicates based on the status of the environment, clauses, and norms. We assume that other predicates may exist whose truth value may be inferred from other sources such as ontologies, or an action model. Each of these sources thus generates a theory, denoted by Γ.

For example, we label the theory generated by the environment as Γ_{Env}. We label the union of all theories as $\mathbf{\Gamma}$.

Formally, a *contract document* contains a set of *clauses* representing *norms* imposed on *agents*. A contract document that has been agreed to by those agents has normative force, and the agents affected by a contract document's norms are the parties to that contract. Since a contract document may be instantiated more than once with different agents playing similar roles, agents are identified within a contract using an indirection mechanism: a contract document imposes norms on a set of *roles*, and agents are associated with these roles before the contract is agreed to.

While not mentioned in this document, additional types of contract documents, such as contract proposals (which represent contract documents to be agreed upon and with no normative weight), may exist. References to contracts in the rest of this paper refer to contract documents which have been agreed upon, and thus carry normative weight.

Structural Definitions

We may now define the structure of norms and contract documents. Since these concepts act upon agents, we begin by defining these entities, as well as roles, which are names referenced to identify the agent upon which a norm acts.

Agent Names and Roles Agents in our framework are left unspecified; we only assume that they are associated with a unique agent name.

A contract role may have one or more parent roles. This means that whenever an agent is assigned to a role, it is also assigned to that role's parent roles, and so assumes the clauses applying to those parents. If a role r_1 is a parent of role r_2, then r_2 may be referred to as the child role of r_1.

Definition 1 (Roles) A role is a constant. We assume that the set of all roles is called $Roles$. Then a role hierarchy definition $RoleHierarchyDefinition$ is a binary relation of the form $(Parent, Child)$ where $Parent, Child \in Roles$. □

For example, the car insurance brokerage contract described earlier contains two roles, that of the *broker*, and that of the *repairer*. One of the clauses obliges the repairer to repair a car, while another assigns a permission to the agent taking on the broker role, allowing it to demand a penalty from the repairer if a car is not fixed by the end of a period specified within the contract.

If we would like to specify that the role of a repairer exists in a contract, and also include the fact that any car repairer also acts as a repairer (i.e. *repairer* is a parent role of *car repairer*), then $(repairer, carRepairer)$ would be contained within $RoleHierarchyDefinition$.

Norms A contract contains a set of clauses, represented by norms. Norms may bind an agent to a certain course of action in all situations, or may only affect an agent when certain activation conditions apply. Similarly, once an agent achieves a certain state of affairs, a norm may no longer apply. Finally, norms affect only a specific set of target agents. A norm thus consists of the following.

- A type identifier, stating whether the norm is an obligation or a permission.
- An activation condition stating when the norm must be instantiated.
- A normative goal or state (condition) used to identify when the norm is violated (in the case of obligations) or what the agent is allowed to do (in the case of permissions).
- An expiration condition used to determine when the norm no longer affects the agent.
- A target, identifying which agents the norm affects.

Norms may be activated, met, and discharged based on various factors including the environment, the status of contracts, and the status of other norms. We assume the existence of $\mathbf{\Gamma}$, a theory (or possibly a set of theories) allowing one to interpret the status of norms[1]. To represent the status of a norm, we define a normative environment theory Γ_{NEnv} below, and assume that it is part of $\mathbf{\Gamma}$.

A norm that may apply to a number of situations is, in a sense, abstract. When a situation to which it applies does arise, the norm is instantiated and exerts a normative force on the agents that are beholden to it. We may thus informally define an abstract norm as a norm that, when the conditions are right, comes into effect (i.e. is instantiated) and only then has normative force over one or more agents. As the name suggests, an instantiated norm is an instantiated abstract norm, which has normative power over a set of agents, until it is discharged.

A group of abstract norms (which, in our case, are the clauses of a contract) is gathered into an abstract norm store. Norms may be represented as a tuple of $wffs$.

Definition 2 (Abstract Norms and Abstract Norm Store) An *Abstract Norm Store*, denoted $\mathcal{ANS}$, consists of a set of abstract norms, each of which is a tuple of the form

$$\langle NormType, NormActivation, NormCondition, NormExpiration, NormTarget\rangle$$

where:
1. $NormType \in$ {obligation, permission}
2. for $N \in \{NormActivation, NormCondition, NormExpiration, NormTarget\}$, N is a *wff* (denoted by ϕ_N) □

We may further divide $NormCondition$ into a state based maintenance condition (labeled $SMaintenanceCondition$) and an action based maintenance condition (labeled ($AMaintenanceCondition$)). A truth value for $NormCondition$ may be computed as the truth value of ($AMaintenanceCondition \wedge SMaintenanceCondition$).

$NormActivation$ is some *wff* ϕ_{NA} which, when entailed by the theory, must be entailed as the fully grounded ϕ'_{NA} in order that the abstract norm can be instantiated and thus

[1]For example, $\mathbf{\Gamma}$ may include references to the environment, an ontology, an action model, and normative environment.

come into force. The substitution of variables S such that $\phi'_{NA} = S(\phi_{NA})$ is then applied to the other components of the abstract norm, thus specifying the instantiated norm.

Instantiating Abstract Norms We now define how abstract norms are instantiated with respect to the domain environment theory and normative environment theory.

An instantiated norm has the same overall form as an abstract norm but its activation condition is grounded, and its remaining parameters are partially grounded using the same grounding as the activation condition.

Definition 3 (Instantiation of Abstract Norms) The abstract norm

$$\langle NormType, NormActivation, NormCondition, NormExpiration, NormTarget\rangle$$

instantiated by the *Environment* Γ_{Env} and *Normative Environment Theory* Γ_{NEnv}, obtains an instantiated norm:

$$\langle NormType, NormActivation', NormCondition', NormExpiration', NormTarget'\rangle$$

where:

- $\mathbf{\Gamma} \vdash NormActivation'$, where $NormActivation'$ is fully grounded such that $NormActivation'$ = $S(NormActivation)$
- $NormCondition'$ = $S(NormCondition)$
- $NormExpiration'$ = $S(NormExpiration)$.
- $NormTarget'$ = $\{X \mid \mathbf{\Gamma} \cup \{NormActivation'\} \cup \{S(NormTarget)\} \vdash X\}$, where $NormTarget' \subseteq AgentNames$

□

Notice that $NormTarget'$ is the set of individuals X to whom the instantiated norm applies. These individuals are identified with reference to (entailed by) the domain environment theory, normative environment[2], and the $NormActivation$ and $NormTarget$ *wffs* that are grounded with respect to the former environments. In the context of a contract clause, the norm's targets are identified by using the $RoleHierarchyDefinition$ relation. Note also that $NormCondition'$ an $NormExpiration'$ may only be partially grounded.

Given a set of abstract norms ANS, together with a $\mathbf{\Gamma}$, we define the set of norms that may be instantiated from $\mathbf{\Gamma}$ as $inst(ANS)$.

Contracts Given the definition of roles, we may specify a contract document as follows:

[2]The normative environment may be used when a target should be identified based on the status of another norm. For example, in the case of a contrary to duty obligation, a penalty must be paid by the agent(s) violating some other norm.

Definition 4 (Contract Document) A Contract Document is a tuple

$$ContractDocument = \langle \mathbf{\Gamma}, CDNorms, CDRoles, CDRoleMapping\rangle$$

where $CDNorms$ is a set of abstract norms. $CDRoles$ is a set of role definitions, and $CDRoleMapping$ maps these role definitions to the set of agents names which are the contract parties within the contract document. We identify a set $ContractParties \subseteq AgentNames$ as those agents named within the contract. This means that the following condition must be satisfied:

$\mathbf{\Gamma} \cup CDRoles \cup CDRoleMapping \vdash X$ where $X \subseteq ContractParties$. □

The qualification on X requires that a contract document only imposes norms on contract parties.

Being an identifiable data item, a contract document, or any element of it, may have additional metadata which may be included in the contract document itself or stored separately. One common piece of metadata associated with most contracts is a contract status. Metadata may be viewed as an additional theory from which the agent can infer information, and is labeled $\Gamma_{metadata}$.

A contract may refer to other contracts, requiring that these additional contracts hold when the referring contract holds. Such additional contracts may also be considered part of the contract's context, and may among other things represent societal regulations imposed on the contracting parties.

Contexts (i.e. ontologies, descriptions and regulations imposed on contract parties) may be shared between multiple contracts. Context provides full meaning to the terms, actions and processes described in the contract.

When an agent reasons about a contract, it makes use of the contract's context. Such an agent is said to be operating within the appropriate context. Agents within a common context share a common vocabulary. This implies that each context has to be associated with a domain ontology defining the meaning of the terms used in the interactions. Therefore, the ontology bound to a context must contain at least all the predicates, roles, role hierarchies, actions and processes that are part of its domain.

However, the ontology may not be enough to express the whole context domain semantics. An extra model is needed for dealing with all the aspects of the domain, especially those that are dynamic. We will call this the *world model*, which contains sets of conditional rules that use predicates and actions from the ontology. The domain ontology and world model also form part of $\mathbf{\Gamma}$.

Contract Proposal Until agreed on by the contract parties, a contract document has no normative weight. At this stage, it is referred to as a contract proposal. Once agreed to, the contract becomes binding, and its norms are then considered to come into effect.

Operational Semantics

So far, we have described the structure of contract documents and norms. Usually, we will be interested not in the

semantics of the documents themselves, but how they affect the contract parties, i.e. how, given the evolution of the environment, various norms are instantiated, fulfilled, violated and discharged.

To do this, we now describe the normative environment theory Γ_{NEnv}. This structure defines predicates that may be used to identify the status of norms as they progress through their lifecycle. A normative environment theory is built around a normative environment, which is itself a (possibly infinite) sequence of normative states $NS_1, NS_2, \ldots$. Each normative state NS_i in the sequence is defined with respect to the overarching theory $\mathbf{\Gamma}$ (which includes Γ_{NEnv}), and a given set of abstract norms ANS.

Each normative state keeps track of four basic events: a)when an abstract norm is instantiated; b)when an instantiated norm expires; c)when a norm's normative condition holds; d)when a norm's normative condition does not hold.

In order to formally define a normative state we first define evaluation of an instantiated norm's $NormCondition$ and $ExpirationCondition$:

Definition 5 (The *holds()* Predicate) Let in be an instantiated norm

$$\langle NormType, NormActivation, NormCondition, \\ NormExpiration, NormTarget\rangle$$

Then, for $N \in \{NormCondition, NormExpiration\}$:

$holds(in, N)$ evaluates to true if $\mathbf{\Gamma} \vdash N'$, where N' is entailed with all variables in N grounded; otherwise $holds(in, N)$ evaluates to *false*. □

Our formal definition of a normative state then identifies those instantiated norms whose normative condition evaluates to true, those whose normative condition evaluates to false, and those whose expiration condition evaluates to true:

Definition 6 (Normative State) Let INS be a set of instantiated norms. A normative state NS, defined with respect to a set INS of instantiated norms, and domain environment theory Γ_{Env} and normative environment theory Γ_{NEnv}, is a tuple of the form:

$$\langle NSTrue, NSFalse, NSExpires\rangle$$

where:

- $NSTrue$ = $\{in \in INS|\ holds(in, NormCondition)$ is true$\}$
- $NSFalse$ = $\{in \in INS|\ holds(in, NormCondition)$ is false$\}$
- $NSExpires$ = $\{in \in INS|\ holds(in, NormExpiration)$ is true$\}$

Since $NSTrue \cup NSFalse \supseteq NSExpires$, it is sufficient to identify the instantiated norms in a normative state, denoted $inst\_norms(NS)$, by the union of those norms whose normative condition evaluates to true, and those, whose normative condition evaluates to false. That is to say:

$$inst\_norms(NS) = NSTrue \cup NSFalse$$ □

Definition 7 (Normative Environment) A normative environment NE is a possibly infinite ordered sequence $NS_1, NS_2, \ldots$ where for $i = 1 \ldots$, we say that NS_i is the normative state *previous* to NS_{i+1}. □

Given a normative state, the subsequent normative state is defined by removal of the expired instantiated norms, addition of new instantiated norms, and checking the norm state of all instantiated norms. We therefore define a minimal set of conditions that a normative environment should satisfy:

Definition 8 (Normative State Semantics) Let $\mathcal{ANS}$ be an abstract norm store, NE the normative environment $NS_1, NS_2, \ldots$, and for $i = 1 \ldots$, $\mathbf{\Gamma_i}$ a set of *wffs* denoting the domain environment associated with NS_i. For $i = 1 \ldots$, let us define the set of potential norms for NS_i as those that:

1. are instantiated in the previous state NS_{i-1} ($inst\_norms(NS_{i-1})$)
2. those in the abstract norm store that are instantiated w.r.t. $\mathbf{\Gamma_i}$ (i.e., $inst(\mathcal{ANS})$ as defined in Definition 3).

and not those that have expired in the previous state, i.e., $NSExpires_{i-1}$.

That is to say, the set of potential norms $PNorms_i$ is defined as follows:

$$PNorms_i = inst\_norms(NS_{i-1}) \cup inst(\mathcal{ANS}) \setminus NSExpires_{i-1}$$

Then $NS_i = \langle NSTrue_i, NSFalse_i, NSExpires_i\rangle$ is defined (as in Definition 6) with respect to the set $PNorms_i$, and theory $\mathbf{\Gamma_i}$. □

We define $NS_0 = \langle NSTrue_0, NSFalse_0, NSExpires_0\rangle$ where $NSTrue_0 = \{\}$, $NSFalse_0 = \{\}$ and $NSExpires_0 = \{\}$

We suggest the following basic set of predicates entailed by Γ_{NEnv}, and in this way characterize how Γ_{NEnv} may be partially specified by the normative environment.[3] In the following definitions we assume a normative environment $\{NS_1, NS_2, \ldots\}$ where $NS_i = \langle NSTrue_i, NSFalse_i, NSExpires_i\rangle$, and $i > 0$. We make use of the *Gödelisation* operator $\lceil . \rceil$ for naming normative states in the object level language. I.e. $\lceil NS_i \rceil$ names normative state NS_i and allows us to use it within *wffs*.

Definition 9 (The *instantiated()* predicate) $\Gamma_{NEnv} \vdash instantiated(\lceil NS_i \rceil, in)$ iff $in \in inst\_norms(NS_i)$ and $(in \notin inst\_norms(NS_{i-1}) \vee in \notin NSExpires_{i-1})$. We define by default $\Gamma_{NEnv} \nvdash instantiated(\lceil NS_0 \rceil, in)$. □

[3] In general, by stating requirement that some first order theory Γ entail $\phi_1, \ldots, \phi_n$, we are effectively providing a partial specification of Γ. In semantic terms, any model for Γ is also model for $\phi_1, \ldots, \phi_n$

Intuitively, $instantiated(NS_i, in)$ holds if the norm in becomes instantiated in NS_i. That is, $instantiated(\lceil NS_i \rceil, in)$ evaluates to true if norm in was instantiated in NS_i, and either was not instantiated in NS_{i-1} or expired in NS_{i-1} (and thus becomes instantiated again in NS_i).

Definition 10 (The *expires()* predicate) $\Gamma_{NEnv} \vdash expires(\lceil NS_i \rceil, in)$ iff $in \in NSExpires_i$. We also define $\Gamma_{NEnv} \nvdash expires(\lceil NS_0 \rceil, in)$. □

The $expires()$ predicate holds if an instantiated norm *in* expired within a specific normative state.

Definition 11 (The *active()* predicate) $\Gamma_{NEnv} \vdash active(\lceil NS_i \rceil, in)$ if and only if $instantiated(\lceil NS_i \rceil, in)$, or else $(in \in inst\_norms(NS_{i-1}) \wedge in \notin NSExpires_{i-1})$. We also define $\Gamma_{NEnv} \nvdash active(\lceil NS_0 \rceil, in)$. □

$active(\lceil NS_i \rceil, in)$ holds if a norm *in* is instantiated within normative state NS_i. This could be because it was instantiated within that state, or because it was instantiated earlier and has not yet expired.

Definition 12 (The *becomesTrue()* predicate) $\Gamma_{NEnv} \vdash becomesTrue(\lceil NS_i \rceil, in)$ iff $in \in NSTrue_i$ and, either $in \in NSFalse_{i-1}$, or $instantiated(\lceil NS_i \rceil, in)$. □

Intuitively, a norm in becomes true in NS_i if its normative condition evaluates to true, and either it was false in state NS_{i-1}, or if not, then in is instantiated in NS_i.

Definition 13 (The *becomesFalse()* predicate) $\Gamma_{NEnv} \vdash becomesFalse(\lceil NS_i \rceil, in)$ iff $in \in NSFalse_i$ and, either $in \in NSTrue_{i-1}$ or $instantiated(\lceil NS_i \rceil, in)$. □

Here, $becomesFalse(\ldots)$ is similar to $becomesTrue(\ldots)$, dealing with falsehood rather than truth. The next two predicates check whether a norm is active and true, respectively false, in some normative state.

Definition 14 (The *isTrue()* predicate) $\Gamma_{NEnv} \vdash isTrue(\lceil NS_i \rceil, in)$ if and only if $becomesTrue(\lceil NS_i \rceil, in)$, or else, $active(\lceil NS_i \rceil, in)$ and $in \in NSTrue_{i-1}$. □

Definition 15 (The *isFalse()* predicate) $\Gamma_{NEnv} \vdash isFalse(\lceil NS_i \rceil, in)$ if and only if $becomesFalse(\lceil NS_i \rceil, in)$, or else, $active(\lceil NS_i \rceil, in)$ and $in \in NSFalse_{i-1}$ □

Definition 16 (Properties of Γ_{NEnv}) $\Gamma_{NEnv} \vdash \neg x$ iff $\Gamma_{NEnv} \nvdash x$. This implies that:

- $\Gamma_{NEnv} \nvdash \bot$.
- $\neg$ is given a negation as failure semantics.

□

Apart from these low level predicates, we may define additional useful predicates. Some of these determine the status of a norm, while others allow access its operation.

Definition 17 (Norm access predicates) Given a norm N with norm type $Type$, activation condition $NormActivation$, expiration condition $NormExpiration$, a norm target set $NormTarget$ a normative condition with a state component $SMaintenanceCondition$ and an action component $AMaintenanceCondition$, the following predicates (which may operate on both abstract and instantiated norms) may be defined:
$type(N, X) = true$ iff $X = Type$, and *false* otherwise.
$normActivation(N, X) = true$ iff $NormActivation$ unifies to X, and *false* otherwise.
$normSCondition(N, X) = true$ iff $SMaintenanceCondition$ unifies to X, and *false* otherwise.
$normACondition(N, X) = true$ iff $AMaintenanceCondition$ unifies to X, and *false* otherwise.
$normExpiration(N, X) = true$ iff $NormExpiration$ unifies to X, and *false* otherwise.
$normTarget(N, A) = true$ iff there is a unification between some element of $NormTarget$ and A. □

We may define the following predicates based on the normative environment. These predicates form a basis for our normative environment theory Γ_{NEnv}:

Definition 18 (the violated() predicate)
$violated(\lceil NS_i \rceil, in) = normType(in, obligation) \wedge isFalse(\lceil NS_i \rceil, in)$ □

The fulfilled predicate checks whether a norm has been fulfilled at a specific point in time

Definition 19 (the fulfilled() predicate)
$fulfilled((\lceil NS_i \rceil, in) = expires(\lceil NS_i \rceil, in) \wedge \neg violated(\lceil NS_i \rceil, in)$
$unfulfilled((\lceil NS_i \rceil, in) = expires(\lceil NS_i \rceil, in) \wedge violated(\lceil NS_i \rceil, in)$ □

We may also be interested in determining whether a norm is a *violation handler*, that is, if it detects and handles the violation of some other clause. We make the simplifying assumption that a violation handler contains only the $violated()$ predicate in its activating condition.

Definition 20 (the violationHandler() predicate)
$violationHandler(N) = normActivation(N, \lceil violated(X, Y) \rceil)$ for any X, Y. □

Finally, we may want to determine which norm (N_1) is the violation handler for another norm (N_2):

Definition 21 (the handlesViolation() predicate)
$handlesViolation(N_1, N_2) = normActivation(N_1, \lceil violated(X, N_2) \rceil)$ for any X □

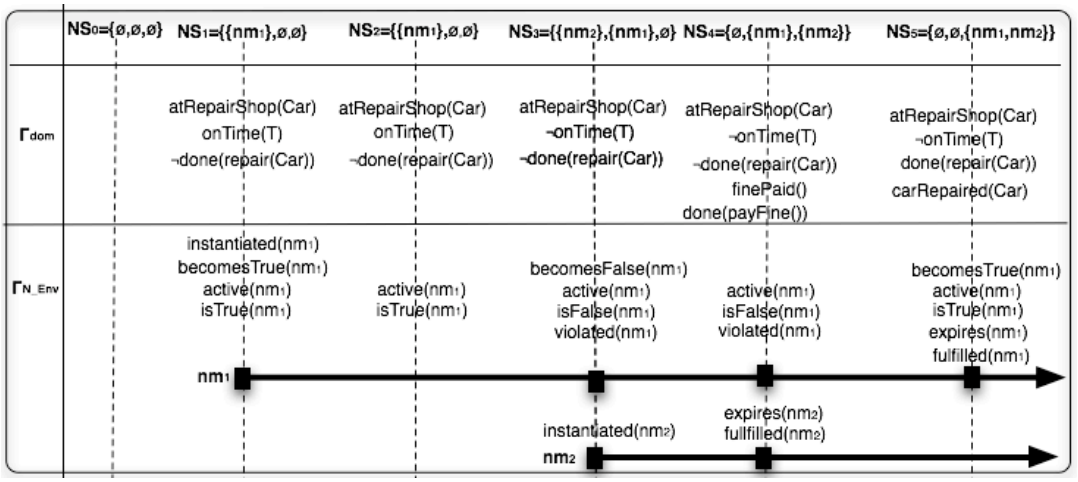

Figure 1: Domain Environment and Normative Environment lifecycle

As discussed later, we intend to make use of these semantics when evaluating the status contracts. Before doing so, we illustrate the use of our framework within an example.

Example

Building on the Car Insurance Brokerage example presented above, we now show how the normative environment theory evolves, together with its attendant normative states. We assume a contract over the repair of a car has been signed between a Repair Company ("Bob Repairs") and the broker (Damage Secure). This contract contains the two clauses[4]:

The first clause consists of norm nm_1 =

$$\langle obligation, atRepairShop(Car), atRepairShop(Car), \\ onTime(T) \vee done(repairCar(Car)), BobRepairs \rangle$$

Where $onTime(T) = before(T, contractStartTime + 7days)$. Here, T represents the current time.
The second clause contains norm nm_2 =

$$\langle obligation, violated(nm_1), done(payFine()), \\ finePaid(), BobRepairs \rangle$$

The first clause expresses the obligation imposed on Bob Repairs to repair a car within the first 7 days of the contract start date, provided the car is at the repair shop. The second expresses the obligation on the Repair Company to pay a fine in case it violates the first obligation. At this stage, it must be noted that the obligation on the Repair Company to repair the car holds even if the seven day deadline has passed. Alternatively, if we wanted to model the obligation in such a way that, if after the seven days pass and the car is still not repaired, a fine should be paid and no repair must be made, then we would have to modify the first $NormExpiration$ attribute to read $carRepaired(Car) \vee after(T, contractStartTime + 7days)$.

[4]Due to space constraints, we do not describe the temporal predicates used in the clauses, instead assuming that they are derived from some standard temporal logic.

Figure 1 shows the values of several of predicates belonging to Γ_{Dom} $(atRepairShop, onTime, done, finePaid)$ and Γ_{NEnv} $(instantiated(), violated(), expires(), ...)$. Some predicates for nm_2, the value of which can easily be inferred, are omitted from the figure, due to limited space.

By default, NS_0 contains empty sets for all its elements. In the next Normative State, NS_1, nm_1's $NormCondition$ becomes true, as the car is at the shop. This event causes nm_1 to become instantiated. The norm's $NormCondition$ element evaluates to true (and thus $nm_1 \in NSTrue_1$) until 7 days from the contract start date have passed. At that point (NS_3), the predicate $before(T, contractStartTime + 7days)$ no longer holds, and $NormCondition$ becomes false (and thus $nm_1 \in NSFalse_1$). By definition, this means that the violated() predicate for nm_1 evaluates to true. This causes the instantiation of the second norm, as seen by its $NormActivation$ parameter. By paying the fine at the next Normative State NS_4, the Repair Company fulfills nm_2's $NormExpiration$ condition, and this brings nm_2 to a $fulfilled()$ state. Finally, we assume that the car is repaired at NS_5, meaning that $violated()$ evaluates to false for nm_1. This means that nm_1 is also fulfilled, as its $NormExpiration$ now evaluates to true.

Discussion and Conclusions

The normative framework we have described fulfills all of the requirements described earlier. Not only are we able to detect whether a violation took place (via the $violation(\ldots)$ predicate), but we may also detect the occurrence of additional *critical states* at which some normative event related state change took place. These critical states correspond to the various predicates described above. Additional, domain dependent critical states may be defined using the information found within the normative environment. Verification of a normative system may be performed by forward simulation over the domain and normative environments.

We assume that any norm aware agent capable of being affected by norms, is associated with its own normative en-

vironment (and resulting normative environment theory). In fully observable environments, each agent's theory would be identical, but in other domains, these theories may diverge. In the context of contracting, we assume that the contract may state which agent's theories should be used when determining whether penalties (or rewards) should be imposed.

Our model does not describe what should occur if an obligation is violated. Instead, we assume that agents make use of the normative model to undertake their own practical reasoning. An agent may determine which norms affect it at any stage, and base its decisions on these.

One interesting aspect of our model (as illustrated by norm nm_1 in the example) is that norms may be violated for a certain period of time, after which they may return to an un-violated state. This is particularly useful when dealing with contracts, as penalties may be assessed over the duration of a violation, with the norm still having normative force over an agent. This differs from the way most deontic theories deal with norms (for example (van der Torre 2003)).

While a large variety of electronic contracting languages exist, many only specify an informal (Milosevic and Dromey 2002), or programming language based (Kollingbaum 2005) semantics. Formal languages, such as LCR (Dignum et al. 2002) have limited expressibility. Our approach of defining a rich contracting language, and then constructing its semantics, is intended to overcome these weaknesses.

The work presented here has been inspired by a number of other researchers. For example, (Dignum 2004) described the use of Landmarks as abstract states which "are defined as a set of propositions that are true in a state represented by the landmark". These landmarks may thus be seen as similar to critical states. The framework described by (Fornara and Colombetti 2007) shares some similarities with our approach. Their focus on sanctions (which, in our model, are implemented more via additional norms) means that they only allow for very specific, predefined normative states, and that violations in their framework may only occur once.

(Farrell et al. 2005) described a predicate based event calculus approach to keeping track of normative state in contracts. However, their work focused on specifying an XML based representation of event calculus, and made use of event calculus primitives to specify their contracts, resulting in a very unwieldy and unrealistic contract representation, and very few norm related predicates. Finally, (Daskalopulu 2000) showed how petri-nets could be used to perform contract monitoring. However, her representation is best suited for those contracts which can be expressed as workflows.

In this paper, we have presented the normative underpinnings of our contract model, showing how norms are represented, and how we can determine their state as the environment changes. We have successfully migrated this approach to the contract level, allowing us to identify the state of a contract (e.g. drafted, active, etc.) at any point in time. IST-CONTRACT intends to create an entire contracting ecosystem, and we are currently using the normative model to define the behavior of various contract-supporting components of the system such as the contract store and contract managers. In the near-term, we intend to see whether we can migrate the semantics described here to additional levels of the framework, for example to specify the form inter-agent protocols. In the long-term, we are interested in examining how norm-conflict mechanisms may best make use of the framework, and are also looking at the effects of partial, and conflicting information on the semantics.

Acknowledgments

This work has been funded mainly by the FP6-034418 IST-CONTRACT project. Javier Vázquez-Salceda's work has been also partially funded by the Ramón y Cajal program of the Spanish Ministry of Education and Science. All the authors would like to thank the CONTRACT project partners for their inputs to this work.

References

Daskalopulu, A. 2000. Modelling Legal Contracts as Processes. In *11th Intl. Conf. and Workshop on Database and Expert Systems Applications*, 1074–1079.

Dignum, V.; Meyer, J. J.; Dignum, F.; and Weigand, H. 2002. Formal specification of interaction in agent societies. In *Proc. of the Second Goddard Workshop on Formal Approaches to Agent Based Systems*, 37–52.

Dignum, V. 2004. *A Model for Organizational Interaction: Based on Agents, Founded in Logic.* Ph.D. Dissertation, Universiteit Utrecht.

Farrell, A. D. H.; Sergot, M.; Salle, M.; and Bartolini, C. 2005. Using the event calculus for tracking the normative state of contracts. *Intl. Journal of Cooperative Information Systems* 4(2–3):99–129.

Fornara, N., and Colombetti, M. 2007. Specifying and enforcing norms in artificial institutions. In *Normative Multi-agent Systems*, number 07122 in Dagstuhl Seminar Proc.

Grosof, B., and Poon, T. C. 2004. SweetDeal: Representing agent contracts with exceptions using semantic web rules, ontologies, and process descriptions. *Intl. Journal of Electronic Commerce* 8(4):61–98.

Jennings, N. R.; Faratin, P.; Lomuscio, A. R.; Parsons, S.; Wooldridge, M.; and Sierra, C. 2001. Automated negotiation: Prospects methods and challenges. *Group Decision and Negotiation* 10(2):199–215.

Kollingbaum, M. 2005. *Norm-governed Practical Reasoning Agents*. Ph.D. Dissertation, University of Aberdeen.

Milosevic, Z., and Dromey, R. G. 2002. On expressing and monitoring behaviour in contracts. In *Proc. of the Sixth Intl. Enterprise Distributed Object Computing Conf.*, 3–14.

van der Torre, L. 2003. Contextual deontic logic: Normative agents, violations and independence. *Annals of Mathematics and Artificial Intelligence* 37(1–2):33–63.

Vasconcelos, W. W.; Kollingbaum, M. J.; and Norman, T. J. 2007. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proc. of the Sixth Intl. Conf. on Autonomous Agents and Multiagent Systems*, 632–639.