Normative Monitoring: Semantics and Implementation

Sergio Alvarez-Napagao[1], Huib Aldewereld[2],
Javier Vázquez-Salceda[1], and Frank Dignum[2]

[1] Universitat Politècnica de Catalunya
{salvarez,jvazquez}@lsi.upc.edu
[2] Universiteit Utrecht
{huib,dignum}@cs.uu.nl

Abstract. The concept of Normative Systems can be used in the scope of Multi-Agent Systems to provide reliable contexts of interactions between agents where acceptable behaviour is specified in terms of norms. Literature on the topic is growing rapidly, and there is a considerable amount of theoretical frameworks for normative environments, some in the form of Electronic Institutions. Most of these approaches focus on regulative norms rather than on substantive norms, and lack a proper implementation of the ontological connection between brute events and institutional facts. In this paper we present a formalism for the monitoring of both regulative (deontic) and substantive (constitutive) norms based on Structural Operational Semantics, its reduction to Production Systems semantics and our current implementation compliant to these semantics.

1 Introduction

In recent years, several researchers have argued that the design of multi-agent systems (MAS) in complex, open environments can benefit from social abstractions in order to cope with problems in coordination, cooperation and trust among agents, problems which are also present in human societies. One of such abstractions is *Normative Systems*. Research in Normative Systems focuses on the concepts of norms and normative environment (which some authors refer to as *institutions*) in order to provide normative frameworks to restrict or guide the behaviour of (software) agents. The main idea is that the interactions among a group of (software) agents are ruled by a set of explicit norms expressed in a computational language representation that agents can interpret. Although some authors only see norms as inflexible restrictions to agent behaviour, others see norms not as a negative, constraining factor but as an aid that guides the agents' choices and reduces the complexity of the environment, making the behaviour of other agents more predictable.

Until recently, most of the work on normative environments works with norm specifications that are static and stable, and which will not change over time. Although this may be good enough from the social (institutional) perspective, it is not appropriate from the agent perspective. During their lifetime, agents may enter and leave several interaction contexts, each with its own normative framework. Furthermore they may be operating in contexts where more than one normative specification applies. So we need

M. De Vos et al. (Eds.): COIN 2010 International Workshops, LNAI 6541, pp. 321–336, 2011.

mechanisms where normative specifications can be added to the agents' knowledge base at run-time and be practically used in their reasoning, both to be able to interpret institutional facts from brute ones (by using constitutive norms to, e.g. decide if killing a person counts as *murder* in the current context) and to decide what ought to be done (by using regulative norms to, e.g. prosecute the murderer). In this paper we propose to use production systems to build a norm monitoring mechanism that can be used both by agents to perceive the current normative state of their environment, and for these environments to detect norm violations and enforce sanctions. Our basic idea is that an agent can configure, at a practical level, the production system at run-time by adding abstract organisational specifications and sets of counts-as rules.

In our approach, the detection of normative states is a passive procedure consisting in monitoring past events and checking them against a set of active norms. This type of reasoning is already covered by the declarative aspect of production systems, so no additional implementation in an imperative language is needed. Using a forward-chaining rule engine, events will automatically trigger the normative state - based on the operational semantics - without requiring a design on *how* to do it.

Having 1) a direct syntactic translation from norms to rules and 2) a logic implemented in an engine consistent with the process we want to accomplish, allows us to decouple normative state monitoring from the agent reasoning. The initial set of rules we have defined is the same for each type of agent and each type of organisation, and the agent will be able to transparently query the current normative state at any moment and reason upon it. Also this decoupling helps building third party/facilitator agents capable of observing, monitoring and reporting normative state change or even enforcing behaviour in the organisation.

In this paper we present a formalism for the monitoring of both regulative (deontic) and substantive (constitutive) norms based on Structural Operational Semantics (Section 2), its reduction to Production Systems semantics (Section 3) and our current implementation compliant to these semantics (Section 4). In Section 5 we compare with other related work and provide some conclusions.

2 Formal Semantics

In this section we discuss the formal semantics of our framework. First, in section 2.1, we give the semantics of institutions as the environment specifying the regulative and constitutive norms. Then, in section 2.2, we describe the details of how this institution evolves over time based on events, and how this impacts the monitoring process. This formalisation will be used in section 3 as basis of our implementation.

Through this paper, we will use as an example the following simplified traffic scenario:

1. A person driving on a street is not allowed to break a traffic convention.
2. In case (1) is violated, the driver must pay a fine.
3. In a city, to exceed 50kmh counts as breaking a traffic convention.

2.1 Preliminary Definitions

Before giving a formal definition of institutions (see Definition 4), we first define the semantics of the regulative and constitutive norms part of that institution (in definitions 1 and 3, respectively).

We assume the use of a predicate based propositional logic language $\mathcal{L}_O$ with predicates and constants taken from an ontology O, and the logical connectives $\{\neg, \vee, \wedge\}$. The set of all possible well-formed formulas of $\mathcal{L}_O$ is denoted as $wff(\mathcal{L}_O)$ and we assume that each formula from $wff(\mathcal{L}_O)$ is normalised in Disjunctive Normal Form (*DNF*). Formulas in $wff(\mathcal{L}_O)$ can be partially grounded, if they use at least one free variable, or fully grounded if they use no free variables.

In this paper we intensively use the concept of variable substitution. We define a substitution instance $\Theta = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, ..., x_i \leftarrow t_i\}$ as the substitution of the terms $t_1, t_2, ..., t_i$ for variables $x_1, x_2, ..., x_i$ in a formula $f \in wff(\mathcal{L}_O)$.

We denote the set of roles in a normative system as the set of constants R, where $R \subset O$, and the set of participants as P, where each participant enacts at least one role according to the ontology O.

As our aim is to build a normative monitoring mechanism that can work at real time, special care has been made to choose a norm language which, without loss of expresiveness, has operational semantics that can then be mapped into production systems. Based in our previous work and experience, our definition of *norm* in an extension of the norm language presented in [12]:

Definition 1. *A norm n is a tuple $n = \langle f_A, f_M, f_\delta, f_D, f_w, w \rangle$, where*

- $f_A, f_M, f_\delta, f_D, f_w \in wff(\mathcal{L}_O)$, $w \in R$,
- f_A, f_M, f_D *respectively represent the activation, maintenance, and deactivation conditions of the norm, f_δ, f_w are the explicit representation of the deadline and target of the norm, and*
- w *is the subject of the norm.*

In order to create an optimal norm monitor it is important to know which norms are active at each point in time, as only those are the ones that have to be traced (inactive norms can be discarded from the monitoring process until they become active again). The *activation condition* f_A specifies when a norm becomes active. It is also the main element in the norm instantiation process: when the conditions in the activating condition hold, the variables are instantiated, creating a new norm instance[1]. The *target* condition f_w describes the state that fulfills the norm (e.g. if one is obliged to pay, the payment being made fulfills the obligation). The *deactivating condition* f_D defines when the norm becomes inactive. Typically it corresponds to the *target* condition (e.g., fulfilling the norm instance deactivates that instance of the norm), but in some cases it also adds conditions to express other deactivating scenarios (e.g., when the norm becomes deprecated). The *maintenance condition* f_M defines the conditions that, when

[1] One main differentiating aspect of our formalisation is that we include variables in the norm representation and we can handle multiple instantiations of the same norm and track them separately.

no longer hold, lead to a violation of the norm. Finally the *deadline* condition f_δ resprents one or several deadlines for the norm to be fulfilled.

An example of a norm for the traffic scenario (*"A person driving on a street is not allowed to break a traffic convention"*) would be formalised as follows

$$\begin{aligned}\mathbf{n1} :=&\langle enacts(X, Driver) \wedge is\_driving(X),\\ &\neg traffic\_violation(X),\\ &\bot,\\ &\neg is\_driving(X),\\ &is\_driving(X) \wedge \neg traffic\_violation(X),\\ &Driver\rangle,\end{aligned}$$

The activating condition states that each time an event appears where an individual enacting the $Driver$ role drives ('$is\_driving$), then a new instance of the norm becomes active; the maintenance condition states that the norm will not be violated while no traffic convention is violated; this norm has no deadline, it is to apply at all times an individual is driving; the norm instance deactivates when the individual stops driving[2]; the target of this norm is that we want drivers not breaking traffic conventions; finally the subject of the norm is someone enacting the $Driver$ role.

It is important to note here that, although our norm representation does not explicitly include deontic operators, the combination of the activation, deactivation and maintenance conditions is as expressive as conditional deontic statements with deadlines as the ones in [3]. It is also able to express unconditional norms and maintenance obligations (i.e. the obligation to keep some conditions holding for a period of time). To show that our representation can be mapped to conditional deontic representations, let us express the semantics of the norm in definition 1 in terms of conditional deontic statements. Given relations between the deadline and maintenance condition (that is, $f_\delta \rightarrow \neg f_M$, since the maintenance condition expresses more than the deadline alone) and between the target and the deactivation condition (i.e., $f_w \rightarrow f_D$, since the deactivation condition specifies that either the norm is fulfilled or something special has happened), we can formalise the norms of Definition 1 as the equivalent deontic expression (using the formalism of [3]): $f_A \rightarrow [O_w(E_w f_w \leq \neg f_M)\,\mathrm{U}\, f_D]$, where $E_a p$ means that agent a sees to it that (*stit*) p becomes true and U is the CTL* until operator. Intuitively, the expression states that after the norm activation, the subject is obliged to see to it that the target becomes true before the maintenance condition is negated (either the deadline is reached or some other condition is broken) until the norm is deactivated (which is either when the norm is fulfilled or has otherwise expired).

Since we are not reasoning about the (effects of) combinations of norms, we will not go into further semantical details here. The semantics presented in this deontic reduction are enough for understanding the monitoring process that is detailed in the remainder of the paper.

A set of norms is denoted as N. We define as *violation handling norms* those norms that are activated automatically by the violation of another norm:

[2] Although the norm is to apply at all times an individual is driving, it is better to deactivate the norm each time the individual stops driving, instead to keep it active, to minimize the number of norm instances the monitor needs to keep track at all times.

Definition 2. *A norm* $n' = \langle f'_A, f'_M, f'_\delta, f'_D, f'_w, w' \rangle$ *is a violation handling norm of* $n = \langle f_A, f_M, f_\delta, f_D, f_w, w \rangle$*, denoted as* $n \rightsquigarrow n'$ *iff* $f_A \wedge \neg f_M \wedge \neg f_D \equiv f'_A$

Violation handling norms are special in the sense that they are only activated when another norm is violated. They are used as *sanctioning norms*, if they are to be fulfilled by the norm violating actor (e.g., the obligation to pay a fine if the driver broke a traffic sign), or as *reparation norms*, if they are to be fulfilled by an institutional actor (e.g. the obligation of the authorities to fix the broken traffic sign).

A norm is defined in an abstract manner, affecting all possible participants enacting a given role. Whenever a norm is active, we will say that there is a *norm instance* $ni = \langle n, \theta \rangle$ for a particular norm n and a substitution instance Θ.

We define the *state of the world* s_t at a specific point of time t as the set of predicates holding at that specific moment, where $s_t \subseteq O$, and we will denote S as the set of all possible states of the world, where $S = \mathcal{P}(O)$. We will call *expansion* $F(s)$ of a state of the world s as the minimal subset of $wff(\mathcal{L}_O)$ that uses the predicates in s in combination of the logical connectives $\{\neg, \vee, \wedge\}$.

One common problem for the monitoring of normative states is the need for an interpretation of brute events as institutional facts, also called constitution of social reality[8]. The use of *counts-as rules* helps solving this problem. Counts-as rules are multi-modal statements of the form $[c](\gamma_1 \rightarrow \gamma_2)$, read as "in context c, γ_1 *counts-as* γ_2". In this paper, we will consider a context as a set of predicates, that is, as a possible subset of a state of the world:

Definition 3. *A counts-as rule is a tuple* $c = \langle \gamma_1, \gamma_2, s \rangle$*, where* $\gamma_1, \gamma_2 \in wff(\mathcal{L}_O)$*, and* $s \subseteq O$*.*

A set of counts-as rules is denoted as C. Although the definition of counts-as in [8] assumes that both γ_1 and γ_2 can be any possible formula, in our work we limit γ_2 to a conjunction of predicates for practical purposes.

Definition 4. *Following the definitions above, we define an* institution *as a tuple of norms, roles, participants, counts-as rules, and an ontology:*
$I = \langle N, R, P, C, O \rangle$

An example of I for the traffic scenario would be formalised as follows:
$\mathbf{N} := \{\langle enacts(X, Driver) \wedge is\_driving(X),$
$\neg traffic\_violation(X), \bot, \neg is\_driving(X),$
$is\_driving(X) \wedge \neg traffic\_violation(X), Driver\rangle,$
$\langle enacts(X, Driver) \wedge is\_driving(X) \wedge traffic\_violation(X),$
$\top,$
$paid\_fine(X), Driver\rangle\}$
$\mathbf{R} := \{Driver\}, \mathbf{P} := \{Person_1\}$
$\mathbf{C} := \{\langle exceeds(D, 50), traffic\_violation(D), is\_in\_city(D)\rangle\}$
$\mathbf{O} := \{role, enacts, is\_driving, is\_in\_city,$
$exceeds, traffic\_violation, is\_driving, paid\_fine,$
$Person_1, role(Driver), enacts(Person_1, Driver)\}$

2.2 Normative Monitor

In this section we present a formalisation of normative monitoring based on Structural Operational Semantics.

From the definitions introduced in section 2.1, a *Normative Monitor* will be composed of the institutional specification, including norms, the current state of the world, and the current normative state.

In order to track the normative state of an institution at any given point of time, we will define three sets: an instantiation set IS, a fulfillment set FS, and a violation set VS, each of them containing norm instances $\{\langle n_i, \Theta_j\rangle, \langle n_{i'}, \Theta_{j'}\rangle, ..., \langle n_{i''}, \Theta_{j''}\rangle\}$. We adapt the semantics for normative states from [11]:

Definition 5. *Norm Lifecycle: Let* $ni = \langle n, \Theta\rangle$ *be a norm instance, where* $n = \langle f_A, f_M, f_D, w\rangle$*, and a state of the world* s *with an expansion* $F(s)$*. The lifecycle for norm instance* ni *is defined by the following normative state predicates:*

$$activated(ni) \Leftrightarrow \exists f \in F(s), \Theta(f_A) \equiv f$$
$$maintained(ni) \Leftrightarrow \exists \Theta', \exists f \in F(s), \Theta'(f_M) \equiv f \wedge \Theta' \subseteq \Theta$$
$$deactivated(ni) \Leftrightarrow \exists \Theta', \exists f \in F(s), \Theta'(f_D) \equiv f \wedge \Theta' \subseteq \Theta$$
$$instantiated(ni) \Leftrightarrow ni \in IS$$
$$violated(ni) \Leftrightarrow ni \in VS$$
$$fulfilled(ni) \Leftrightarrow ni \in FS$$

where IS *is the instantiation set,* FS *is the fulfillment set, and* VS *is the violation set, as defined above.*

For instance, for norm $n1$, the lifecycle is represented in Figure 1. The maintained state is not represented as it holds in both the activated and fulfilled states. The deactivated state is also not depicted because it corresponds in this case to the Fulfilled state.

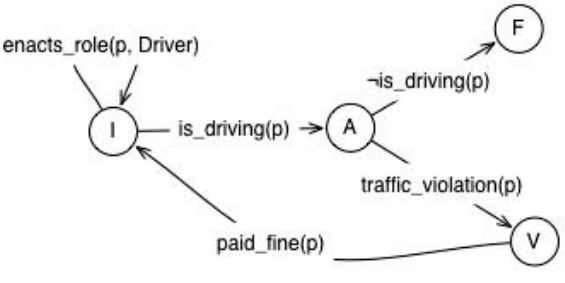

Fig. 1. Lifecycle for norm *n1* in the traffic scenario: (I)nactive, (A)ctivated, (V)iolated, (F)ulfilled

Definition 6. *A Normative Monitor* M_I *for an institution* I *is a tuple* $M_I = \langle I, S, IS, VS, FS\rangle$ *where*

- $I = \langle N, R, P, C, O\rangle$,
- $S = \mathcal{P}(O)$,
- $IS = \mathcal{P}(N \times S \times Dom(S))$,
- $VS = \mathcal{P}(N \times S \times Dom(S))$, *and*
- $FS = \mathcal{P}(N \times S \times Dom(S))$.

Event processed:

$$\frac{e_i = \langle \alpha, p \rangle}{\langle\langle\langle i, s, is, vs, fs\rangle, e_i\rangle, e_{i+1}\rangle \rhd \langle\langle i, s \cup \{p\}, is, vs, fs\rangle, e_{i+1}\rangle} \quad (1)$$

Counts-as rule activation:

$$\frac{\exists \Theta, \exists f \in F(s), \exists \langle \gamma_1, \gamma_2, s_i \rangle \in C, s_i \subseteq s \wedge \Theta(\gamma_1) \equiv f \wedge \Theta(\gamma_2) \notin s}{\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s \cup \{\Theta(\gamma_2)\}, is, vs, fs\rangle, e\rangle} \quad (2)$$

Counts-as rule deactivation:

$$\frac{\exists \Theta, \exists f \in F(s), \exists \langle \gamma_1, \gamma_2, s_i \rangle \in C, s_i \not\subseteq s \wedge \Theta(\gamma_1) \equiv f \wedge \Theta(\gamma_2) \in s}{\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s - \{\Theta(\gamma_2)\}, is, vs, fs\rangle, e\rangle} \quad (3)$$

Norm instantiation:

$$\frac{\exists n = \langle f_A, f_M, f_D, w\rangle \in N \wedge \neg \exists n' \in N, n' \rightsquigarrow n \wedge \langle n, \Theta\rangle \notin is \wedge \exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_A)}{\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s, is \cup \{\langle n, \Theta\rangle\}, vs, fs\rangle, e\rangle} \quad (4)$$

Norm instance violation:

$$\frac{\begin{array}{c}\exists n = \langle f_A, f_M, f_D, w\rangle \in N \wedge \langle n, \Theta'\rangle \in is \wedge \langle n, \Theta'\rangle \notin vs \wedge \\ \neg(\exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_M) \wedge \Theta \subseteq \Theta') \wedge NR = \bigcup_{n \rightsquigarrow n'} \langle n', \Theta'\rangle\end{array}}{\begin{array}{c}\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s, (is - \{\langle n, Theta'\rangle\}) \cup NR, \\ vs \cup \{\langle n, \Theta'\rangle\}, fs\rangle, e\rangle\end{array}} \quad (5)$$

Norm instance fulfilled:

$$\frac{\exists n = \langle f_A, f_M, f_D, w\rangle \in N \wedge \langle n, \Theta'\rangle \in is \wedge \exists \Theta, \exists f' \in F(s), f' \equiv \Theta(f_D) \wedge \Theta \subseteq \Theta'}{\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s, is - \{\langle n, \Theta'\rangle\}, vs, fs \cup \langle n, \Theta'\rangle\rangle, e\rangle} \quad (6)$$

Norm instance violation repaired:

$$\frac{\exists n, n' \in N \wedge n \rightsquigarrow n' \wedge \langle n, \Theta\rangle \in vs \wedge n \rightsquigarrow n' \wedge \langle n', \Theta\rangle \in fs}{\langle\langle\langle N, R, P, C, O\rangle, s, is, vs, fs\rangle, e\rangle \rhd \langle\langle\langle N, R, P, C, O\rangle, s, is, vs - \{\langle n, \Theta\rangle\}, fs\rangle, e\rangle} \quad (7)$$

Fig. 2. Inference rules for the transition relation $\rhd$

The set Γ of possible configurations of a Normative Monitor M_I is $\Gamma = I \times S \times IS \times VS \times FS$.

However, the definition above does not take into account the dynamic aspects of incoming events affecting the state of the world through time. To extend our model we will assume that there is a continuous, sequential stream of events received by the monitor:

Definition 7. *An event e is a tuple $e = \langle \alpha, p\rangle$, where*

- $\alpha \in P$[3]*, and*
- $p \in S$ *and is fully grounded.*

We define E as the set of all possible events, $E = \mathcal{P}(P \times S)$

[3] α is considered to be the asserter of the event. Although we are not going to use this element in this paper, its use may be of importance when extending or updating this model.

Definition 8. *The* Labelled Transition System *for a Normative Monitor M_I is defined by $\langle \Gamma, E, \rhd \rangle$ where*

- *E is the set of all possible events $e = \langle \alpha, p \rangle$*
- *$\rhd$ is a* transition relation *such that $\rhd \subseteq \Gamma \times E \times \Gamma$*

The inference rules for the transition relation $\rhd$ are depicted in Figure 2.

3 Monitoring with Production Systems

In our approach, practical normative reasoning is based on a production system with an initial set of rules implementing the operational semantics described in Section 2.2. Production systems are composed of a set of rules, a working memory, and a rule interpreter or engine [2]. Rules are simple conditional statements, usually of the form *IF a THEN b*, where *a* is usually called left-hand side (*LHS*) and *b* is usually called right-hand side (*RHS*).

3.1 Semantics of Production Systems

In this paper we use a simplified version of the semantics for production systems introduced in [1].

Considering a set $\mathcal{P}$ of predicate symbols, and an infinite set of variables $\mathcal{X}$, where a fact is a ground term, $f \in \mathcal{T}(\mathcal{P})$, and $\mathcal{WM}$ is the *working memory*, a set of facts, a production rule is denoted `if` p, c `remove` r `add` a, or

$$p,\ c \Rightarrow r,\ a,$$

consisting of the following components:

- A set of positive or negative patterns $p = p^+ \cup p^-$ where a pattern is a term $p_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and a negated pattern is denoted $\neg p_i$. p^- is the set of all negated patterns and p^+ is the set of the remaining patterns
- A proposition c whose set of free variables is a subset of the pattern variables: $Var(c) \subseteq Var(p)$.
- A set r of terms whose instances could be intuitively considered as intended to be removed from the working memory when the rule is fired, $r = \{r_i\}_{i \in I_r}$, where $Var(r) \subseteq Var(p^+)$.
- A set a of terms whose instances could be intuitively considered as intended to be added to the working memory when the rule is fired, $a = \{a_i\}_{i \in I_a}$, where $Var(a) \subseteq Var(p)$.

Definition 9. *A set of positive patterns p^+* matches *to a set of facts $\mathcal{S}$ and a substitution σ iff $\forall p \in p^+, \exists t \in \mathcal{S}, \sigma(p) = t$. Similarly, a set of negative patterns p^-* dismatches *a set of facts $\mathcal{S}$ iff $\forall \neg p \in p^-, \forall t \in \mathcal{S}, \forall \sigma, \sigma(p) \neq t$.*

A production rule $p \Rightarrow r, a$ is $(\sigma, \mathcal{WM}')$-fireable *on a working memory $\mathcal{WM}$ when p^+ matches with $\mathcal{WM}'$ and p^- dismatches with $\mathcal{WM}$, where $\mathcal{WM}'$ is a minimal subset of $\mathcal{WM}$, and $\mathcal{T} \models \sigma(c)$.*

Definition 10. *The* application *of a $(\sigma, \mathcal{WM}')$-fireable rule on a working memory $\mathcal{WM}$ leads to the new working memory $\mathcal{WM}'' = (\mathcal{WM} - \sigma(r)) \cup \sigma(a)$.*

Definition 11. *A general production system $\mathcal{PS}$ is defined as $\mathcal{PS} = \langle \mathcal{P}, \mathcal{WM}_0, \mathcal{R} \rangle$, where $\mathcal{R}$ is a set of production rules over $\mathcal{H} = \langle \mathcal{P}, \mathcal{X} \rangle$.*

3.2 Reduction

In order to formalise our Normative Monitor as a production system, we will need to define several predicates to bind norms to their conditions: *activation*, *maintenance*, *deactivation*, and to represent normative state over norm instances: *violated*, *instantiated*, and *fulfilled*. We will also use a predicate for the arrival of events: *event*, and a predicate to represent the fact that a norm instance is a violation handling norm instance of a violated instance: *repair*. For the handling of the DNF clauses, we will use the predicates *holds* and *has_clause*.

Definition 12. *The set of predicates for our production system, for an institution $I = \langle N, R, P, C, O \rangle$, is:*

$$\mathcal{P}_I := O \cup \{activated, maintained, deactivated, violated, instantiated, fulfilled, event, repair, holds, has\_clause, countsas\}$$

The initial working memory $\mathcal{WM}_0$ should include the institutional specification in the form of the formulas included in the counts-as rules and the norms in order to represent the possible instantiations of the predicate $holds$, through the use of the predicate has_clause.

First of all, we need to have the bindings between the norms and their formulas available in the working memory. For each norm $n = \langle f_A, f_M, f_D, w \rangle$, these bindings will be:

$$\mathcal{WM}_n := \{activation(n, f_A), maintenance(n, f_M), deactivation(n, f_D)\}$$

As we assume the formulas from $wff(\mathcal{L}_O)$ to be in DNF form:

Definition 13. *We can interpret a formula as a set of conjunctive clauses $f = \{f_1, f_2, ..., f_N\}$, of which only one of these clauses f_i holding true is necessary for f holding true as well:*

$$r^h := has\_clause(f, f') \wedge holds(f', \Theta) \Rightarrow \emptyset, \{holds(f, \Theta)\}$$

For example, if $f = (p_1(x) \wedge p_2(y) \wedge ... \wedge p_i(z)) \vee ... \vee (q_1(w) \wedge q_2(x) \wedge ... \wedge q_j(y))$, then the initial facts to be in $\mathcal{WM}_0$ will be:

$$\mathcal{WM}_0 := \bigcup_{f' \in f} has\_clause(f, f') = \{has\_clause(f, f_1), ..., has\_clause(f, f_2)\}$$

Also, we have to include the set of repair norms by the use of the predicate $repair$, and the counts-as definitions by the use of the predicate $countsas$.

Definition 14. *The initial working memory $\mathcal{WM}_I$ for an institution $I{=}\langle N, R, P, C, O\rangle$ is:*

$$\mathcal{WM}_I := \bigcup_{n \rightsquigarrow n'}^{n \in N} repair(n, n') \cup \bigcup_{n=\langle f_A, f_M, f_D, w\rangle \in N} (\mathcal{WM}_n \cup \mathcal{WM}_{f_A} \cup \mathcal{WM}_{f_M} \cup \mathcal{WM}_{f_D}) \cup \bigcup_{c=\langle \gamma_1, \gamma_2, s\rangle \in C} (\{countsas(\gamma_1, \gamma_2, s)\} \cup \mathcal{WM}_{\gamma_1} \cup \mathcal{WM}_s)$$

The rule for the detection of a holding formula is defined as $r_f^{hc} = \lceil f \rceil \Rightarrow \emptyset, \{holds(f, \sigma)\}$, where we denote as $\lceil f \rceil$ the propositional content of a formula $f \in wff(\mathcal{L}_O)$ which only uses predicates from O and the logical connectives $\neg$ and $\wedge$, and σ as the substitution set of the activation of the rule. Following the previous example:

$$r_{f_1}^{hc} = p_1(x) \wedge p_2(y) \wedge ... \wedge p_i(z) \Rightarrow \emptyset, \{holds(f_1, \{x, y, z\})\}$$
$$r_{f_2}^{hc} = q_1(w) \wedge q_2(x) \wedge ... \wedge q_i(y) \Rightarrow \emptyset, \{holds(f_2, \{w, x, y\})\}$$

Similarly as in Definition 14:

Definition 15. *The set of rules R_I^{hc} for detection of holding formulas for an institution $I = \langle N, R, P, C, O\rangle$ is:*

$$R_I^{hc} := \bigcup_{n=\langle f_A, f_M, f_D, w\rangle \in N} (\bigcup_{f \in \{f_A, f_M, f_D\}} r_f^{hc}) \cup \bigcup_{c=\langle \gamma_1, \gamma_2, s\rangle \in C} (\bigcup_{f \in \gamma_1} r_f^{hc})$$

By using the predicate $holds$ as defined above, we can translate the inference rules from Section 2.2. Please note that the rules are of the form $p, c \Rightarrow r, a$ as shown in Section 3.1. However, as we only need the c part to create a constraint proposition in the rules for norm instance violation and fulfillment, c is omitted except for these two particular cases.

Definition 16. *Translated rules (see Figure 2)*

Rule for event processing (1):
$r^e = event(\alpha, p) \Rightarrow \emptyset, \{\lceil p \rceil\}$

Rule for counts-as rule activation (2):
$r^{ca} = countsas(\gamma_1, \gamma_2, c) \wedge holds(\gamma_1, \Theta) \wedge holds(c, \Theta') \wedge \neg holds(\gamma_2, \Theta)$
$\Rightarrow \emptyset, \{\Theta(\lceil \gamma_2 \rceil)\}$

Rule for counts-as rule deactivation (3):
$r^{cd} = countsas(\gamma_1, \gamma_2, c) \wedge holds(\gamma_1, \Theta) \wedge \neg holds(c, \Theta') \wedge holds(\gamma_2, \Theta)$
$\Rightarrow \{\Theta(\lceil \gamma_2 \rceil)\}, \emptyset$

Rule for norm instantiation (4):
$r^{ni} = activation(n, f) \wedge holds(f, \Theta) \wedge \neg instantiated(n, \Theta) \wedge \neg repair(n', n)$
$\Rightarrow \emptyset, \{instantiated(n, \Theta)\}$

Rule for norm instance violation (5):
$r^{nv} = instantiated(n, \Theta) \wedge maintenance(n, f) \wedge \neg holds(f, \Theta') \wedge repair(n, n'),$
$\forall \Theta', \Theta' \subseteq \Theta$
$\Rightarrow \{instantiated(n, \Theta)\}, \{violated(n, \Theta), instantiated(n', \Theta)\}$

Rule for norm instance fulfillment (6):
$r^{nf} = deactivation(n, f) \wedge instantiated(n, \Theta) \wedge subseteq(\Theta', \Theta) \wedge holds(f, \Theta'),$
$\Theta' \subseteq \Theta$
$\Rightarrow \{instantiated(n, \Theta)\}, \{fulfilled(n, \Theta)\}$
Rule for norm instance violation repaired (7):
$r^{nr} = violated(n, \Theta) \wedge repair(n, n') \wedge fulfilled(n', \Theta')$
$\Rightarrow \{violated(n, \Theta)\}, \emptyset$

Definition 17. *Following Definitions 13, 15 and 16, the set of rules for an institution* $I = \langle N, R, P, C, O \rangle$ *are:*

$$\mathcal{R}_I := R_I^{hc} \cup \{r^h, r^e, r^{ca}, r^{cd}, r^{ni}, r^{nv}, r^{nf}, r^{nr}\}$$

Definition 18. *The production system* $\mathcal{PS}_I$ *for an institution* I *will be, from Definitions 12, 14 and 17:*

$$\mathcal{PS}_I := \langle \mathcal{P}_I, \mathcal{WM}_I, \mathcal{R}_I \rangle$$

4 Implementation

A prototype of our normative reasoner has been implemented as a DROOLS program. DROOLS is an open-source Object-Oriented rule engine for declarative reasoning in Java [14]. Its rule engine is an implementation of the forward chaining inference Rete algorithm [4]. The use of Java objects inside the rule engine allows for portability and an easier communication of concepts with the reasoning of agents coded in Java.

In DROOLS we can represent facts by adding them to the knowledge base as objects of the class *Predicate*. Predicates are dynamically imported from standardised Description Logic OWL-DL ontologies into Java objects using the tool *OWL2Java*[17], as

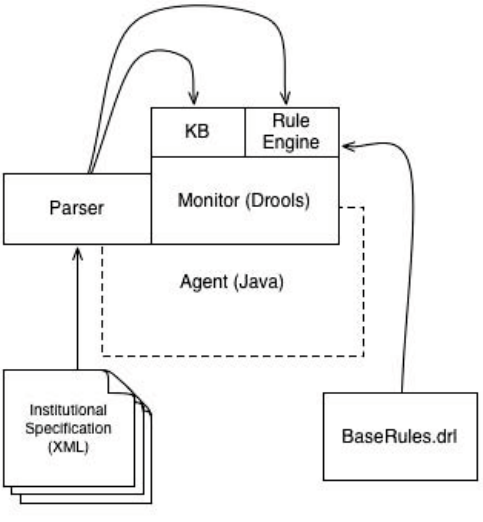

Fig. 3. Architecture of the DROOLS implementation

```
rule "holds"
when
  HasClause(f : formula, f2 : clause)
  Holds(formula == f2, theta : substitution)
then
  insertLogical(new Holds(f, theta));
end

rule "event processed"
when
  Event(a : asserter, p : content)
then
  insertLogical(p);
end

rule "counts-as activation"
when
  CountsAs(g1 : gamma1, g2 : gamma2, s : context)
  Holds(formula == g1, theta : substitution)
  Holds(formula == s, theta2 : substitution)
  not Holds(formula == g2, substitution == theta)
then
  Formula f;

  f = g2.substitute(theta);
  insert(f);
end

rule "counts-as deactivation"
when
  CountsAs(g1 : gamma1, g2 : gamma2, s : context)
  Holds(formula == g1, theta : substitution)
  not Holds(formula == s, theta2 : substitution)
  Holds(formula == g2, substitution == theta)
  f : Formula(content == g2, grounding == theta)
then
  retract(f);
end

rule "norm instantiation"
when
  Activation(n : norm, f : formula)
  Holds(formula == f, theta : substitution)
  not Instantiated(norm == n, substitution == theta)
  not Repair(n2 : norm, repairNorm == n)
then
  insert(new Instantiated(n, theta));
end

rule "norm instance violation"
when
  ni : Instantiated(n : norm, theta : substitution)
  Maintenance(norm == n, f : formula)
  not (SubsetEQ(theta2 : subset, superset == theta)
  and Holds(formula == f, substitution == theta2))
  Repair(norm == n, n2 : repairNorm)
then
  retract(ni);
  insert(new Violated(n, theta));
  insert(new Instantiated(n2, theta));
end

rule "norm instance fulfillment"
when
  Deactivation(n : norm, f : formula)
  ni : Instantiated(norm == n, theta : substitution)
  SubsetEQ(theta2 : subset, superset == theta)
  Holds(formula == f, substitution == theta2)
then
  retract(ni);
  insert(new Fulfilled(n, theta));
end

rule "norm instance violation repaired"
when
  ni : Violated(n : norm, theta : substitution)
  Repair(norm == n, n2 : repairNorm)
  Fulfilled(norm == n2, substitution == theta)
then
  retract(ni);
end

rule "subseteq"
when
  Holds(f : formula, theta : substitution)
  Holds(f2 : formula, theta2 : substitution)
  eval(theta.containsAll(theta2))
then
  insertLogical(new SubsetEQ(theta2, theta));
end
```

Fig. 4. Translation of base rules to DROOLS

subclasses of a specifically designed *Predicate* class. The following shows an example of the insertion of $enacts\_role(p, Driver)$ into the knowledge base to express that p (represented as object p of the domain and instantiating a participant) is in fact enacting the role *driver*:

```
ksession.insert(new Enacts(p, Driver.class));
```

DROOLS programs can be initialised with a rule definition file. However, its working memory and rule base can be modified at run-time by the Java process that is running the rule engine. We take advantage of this by keeping a fixed base, which is a file with fixed contents implementing the rules from Definition 13 and 16, which are independent of the institution, and having a parser for institutional definitions that will feed the rules from Definition 15, which are dependent on the institution (see Figure 3). The institutional definitions we currently use are based on an extension of the XML language presented in [12].

The base rules (see Definitions 13 and 16) has been quite straightforward and the translation is almost literal. The contents of the reusable DROOLS file is shown in Figure 4. The last rule of the Figure is the declarative implementation of the predicate *SubsetEQ* to represent the comparison of substitutions instances $\Theta \subseteq \Theta'$, needed for

```
rule "N1_activation_1"
when
  n : Norm(id == "N1")
  Activation(norm == n, f : formula)
  Enacts(X : p0, p1 == "Driver")
  IsDriving(p0 == X)
then
  Set<Value> theta = new Set<Value>();
  theta.add(new Value("X", X));
  insert(new Holds(f.getClause(0), theta));
end

rule "C1_1"
when
  c : CountsAs(g1 : gamma1)
  Exceeds(D : p0, 50 : p1)
then
  Set<Value> theta = new Set<Value>();
  theta.add(new Value("D", D));
  insert(new Holds(g1.getClause(0), theta));
end
```

Fig. 5. Rules for the traffic scenario

```
ksession.insert(norm1);
ksession.insert(norm2);
ksession.insert(new Repair(norm1, norm2));
ksession.insert(new Activation(norm1, fn1a));
ksession.insert(new Maintenance(norm1, fn1m));
ksession.insert(new Deactivation(norm1, fn1d));
ksession.insert(new HasClause(fn1a, fn1a1));
ksession.insert(new HasClause(fn1m, fn1m1));
ksession.insert(new HasClause(fn1d, fn1d1));
/*          ...same for norm2...          */
ksession.insert(new CountsAs(c1g1, c1g2, c1s));
ksession.insert(new HasClause(c1g1, c1g11));
ksession.insert(new HasClause(c1g2, c1g21));
ksession.insert(new HasClause(c1s, c1s1));
```

Fig. 6. Facts for the traffic scenario

the cases of norm instance violation and fulfillment. In our implementation in *Drools*, substitution instances are implemented as *Set<Value>* objects, where *Value* is a tuple $\langle$*String, Object*$\rangle$.

The rest of the rules (see Definitions 15) are automatically generated from the institutional specifications and inserted into the DROOLS rule engine. An example of two generated rules for the traffic scenario is shown in Figure 5.

The initial working memory is also automatically generated by inserting objects (facts) into the DROOLS knowledge base following Definition 14. An example for the traffic scenario is also shown in Figure 6. Please note that this is not an output of the parser, but a representation of what it would execute at run-time.

5 Conclusions and Related Work

The implementation of rule-based norm operationalisation has already been explored in previous research. Some approaches [13,15] directly define the operationalisation of

the norms as rules of a specific language, not allowing enough abstraction to define norms at a high level to be operationalised in different rule engine specifications. [5] introduces a translation scheme, but it is bound to Jess by using specific constructs of this language and it does not support constitutive norms. Other recent approaches like [6] define rule-based languages with expressive constructs to model norms, but they are bound to a proper interpreter and have no grounding on a general production system, requiring the use of an intentionally crafted or modified rule engine. For example, in [7,9], obligations, permissions and prohibitions are asserted as facts by the execution of the rules, but the actual monitoring is out of the base rule engine used.

[16] introduces a language for defining an organisation in terms of roles, norms, and sanctions. This language is presented along with an operational semantics based on transition rules, thus making its adoption by a general production system straightforward. Although a combination of counts-as rules and sanctions is used in this language, it is not expressive enough to support regulative norms with conditional deontic statements.

We solve these issues by combining a normative language [12] with a reduction to a representation with clear operational semantics based on the framework in [11] for deontic norms and the use of counts-as rules for constitutive norms. The formalism presented in this paper uses logic conditions that determine the state of a norm (active, fulfilled, violated). These conditions can be expressed in propositional logic at the moment and can be directly translated into *LHS* parts of rules, with no special adaptation needed. The implementation of the operational semantics in a production system to get a practical normative reasoner is thus straightforward. This allows agents for dynamically changing its institutional context at any moment, by *feeding* the production system with a new abstract institutional specification.

Our intention is not to design a general purpose reasoner for normative agents, but a practical reasoner for detecting event-driven normative states. This practical reasoner can then be used as a component not only by normative agents, but also by monitors or managers. Normative agents should deal with issues such as planning and future possibilities, but monitors are focused on past events. For such a practical reasoner, the expressivity of actions languages like $C+$ is not needed, and a simple yet efficient solution is to use production systems, as opposed to approaches more directly related to offline verification or model checking, such as [10].

Mere syntactical translations are usually misleading in the sense that rule language specific constructs are commonly used, constraining reusability [13,5,7]. However, as we have presented in this paper a reduction to a general version of production system semantics, any rule engine could fit our purposes. There are several production system implementations available, some widely used by the industry, such as JESS, DROOLS, SOAR or PROVA. In most of these systems rules are syntactically and semantically similar, so switching from one to the other would be quite simple. As production systems dynamically compile rules to efficient structures, they can be used as well to validate and verify the consistency of the norms. As opposed to [7,9], our reduction ensures that the whole monitoring process is carried out entirely by a general production system, thus effectively decoupling normative state detection and agent reasoning.

DROOLS is an open-source powerful suite supported by JBoss, the community, and the industry, and at the same time it is lightweight enough while including key features

that we are or will be using in future work. As an advantage over other alternatives, it includes features relevant to our topic, e.g. event processing, workflow integration. Its OO approach makes it easy to be integrated with imperative code (Java), and OWL-DL native support is expected in a short time.

The monitoring system is available at `http://sf.net/projects/ict-alive` under a GPL license. This implementation is currently being used in use cases with large amounts of events, and we expect to present empirical results of performance as well as an analysis of the algorithmic complexity. A topic that we will cover in more detail in future publications, due to the complexity of the issue on its own and lack of space in this paper, is the addition, modification and removal of normative contexts at run-time. Finally, due to the fact that semantics based on propositional logic can be limiting at a practical level for norm expressivity, as future work we are extending the semantics in order to support, at least, first-order logic norm conditions.

Acknowledgements

This work has been partially supported by the FP7-215890 ALIVE project. J. Vázquez-Salceda's work has been also partially funded by the Ramón y Cajal program of the Spanish Ministry of Education and Science.

References

1. Cirstea, H., Kirchner, C., Moossen, M., Moreau, P.E.: Production Systems and Rete Algorithm Formalisation. Tech. Rep. ILOG, INRIA Lorraine, INRIA Rocquencourt, Manifico (2004)
2. Davis, R., King, J.: An overview of production systems. Tech. rep., Stanford Artificial Intelligence Laboratory, Report No. STAN-CS-75-524 (1975)
3. Dignum, F., Broersen, J., Dignum, V., Meyer, J.J.: Meeting the Deadline: Why, When and How. In: Hinchey, M.G., Rash, J.L., Truszkowski, W.F., Rouff, C.A. (eds.) FAABS 2004. LNCS (LNAI), vol. 3228, pp. 30–40. Springer, Heidelberg (2004)
4. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (1982)
5. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing norms in electronic institutions. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, Netherlands, pp. 667–673 (2005)
6. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rulebased programming of norms for electronic institutions. Autonomous Agents and Multi-Agent Systems 18(1), 186–217 (2009)
7. Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems 14(2-3), 181–216 (2005)
8. Grossi, D.: Designing invisible handcuffs: Formal investigations in institutions and organizations for multi-agent systems. Thesis, Universiteit Utrecht (2007)
9. Hübner, J.F., Boissier, O., Bordini, R.H.: A normative organisation programming language for organisation management infrastructures. In: Padget, J., Artikis, A., Vasconcelos, W., Stathis, K., da Silva, V.T., Matson, E., Polleres, A. (eds.) COIN@AAMAS 2009. LNCS, vol. 6069, pp. 114–129. Springer, Heidelberg (2010)

10. Kyas, M., Prisacariu, C., Schneider, G.: Run-time monitoring of electronic contracts. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 397–407. Springer, Heidelberg (2008)
11. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a formalisation of electronic contracting environments. In: Hübner, J.F., Matson, E., Boissier, O., Dignum, V. (eds.) COIN@AAMAS 2008. LNCS, vol. 5428, pp. 156–171. Springer, Heidelberg (2009)
12. Panagiotidi, S., Vázquez-Salceda, J., Alvarez-Napagao, S., Ortega-Martorell, S., Willmott, S., Confalonieri, R., Storms, P.: Intelligent Contracting Agents Language. In: Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems (BRMAS 2008) at AISB 2008, Aberdeen, Scotland, vol. 1, p. 49 (2008)
13. Paschke, A., Dietrich, J., Kuhla, K.: A Logic Based SLA Management Framework. In: Proceedings of the 4th Semantic Web Conference (ISWC 2005), Galway, Ireland, pp. 68–83 (2005)
14. Proctor, M., Neale, M., Frandsen, M., Griffith Jr., S., Tirelli, E., Meyer, F., Verlaenen, K.: Drools documentation. JBoss (2008)
15. Strano, M., Molina-Jimenez, C., Shrivastava, S.: A rule-based notation to specify executable electronic contracts. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2008. LNCS, vol. 5321, pp. 81–88. Springer, Heidelberg (2008)
16. Tinnemeier, N., Dastani, M., Meyer, J.J.: Roles and norms for programming agent organizations. In: Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, vol. 1, pp. 121–128 (2009)
17. Zimmermann, M.: OWL2Java (2009), `http://www.incunabulum.de/projects/it/owl2java`