Norm-Oriented Programming Language for Electronic Institutions

Andrés García-Camino[1], J. A. Rodríguez-Aguilar[1],
Carles Sierra[1], and Wamberto Vasconcelos[2]

[1] IIIA-CSIC, Campus UAB 08193 Bellaterra Spain
{andres,jar,sierra}@iiia.csic.es

[2] Dept. of Computing Science, Univ. of Aberdeen, AB24 3UE, UK,
wvasconcelos@acm.org

Abstract. Norms constitute a powerful coordination mechanism among heterogeneous agents [27, 4]. This work proposes a means to specify and control the normative dynamics of societies of software agents. For this we introduce a language with which one can explicitly manage the normative positions of agents [21] and distinct deontic notions and their relationships can be captured. This language is conceived as a machine language to facilitate norm-oriented programming and to found higher-level normative languages. We provide a model-theoretic semantics to our formalism, as well as an operational semantics. Furthermore, we show that our rule-based language captures the expressiveness of a wide range of normative models and systems in the literature.

1 Introduction

At the beginning of the multi-agent system (MAS) paradigm, the behaviour of agents was regulated assuming that the system is closed, imposing a certain architecture and specification language in the design of agents which can be easily verified not to violate any specification requirement.

One of the major challenges in MAS research is the design and implementation of *open* multi-agent systems where agents can be specified with different architectures and different specification languages by several, maybe distrusted, designers [12]. Norms can be used for this purpose as a mean to regulate the observable behaviour of agents as they interact in pursuit of their goals [7, 17]. There is a wealth of socio-philosophical and logic-theoretical investigation on the subject of norms (*e.g.*, [5, 21–23]). But recently more attention has being paid to more pragmatic and implementational aspects of norms, that is, how norms can be given a computational interpretation and how norms can be factored in the design and execution of MASs (e.g. [10]).

And yet, we believe that there is a need for norm-oriented programming languages that support the implementation of normative systems. We try to make headway along this direction by introducing a language to specify agents' normative positions and manage their changes as agents interact. A normative position [21] is the "social burden" associated with individual agents, that is, their obligations, permissions and prohibitions. Depending on what agents do, their normative positions may change – for instance, permissions/prohibitions can be revoked or obligations, once fulfilled, may be

removed. We present a language that acts as a "machine language" for norms on top of which higher-level languages may eventually be accommodated. This language can represent distinct flavours of deontic notions and relationships. Although our language is rule-based, we achieve greater flexibility, expressiveness and precision than production systems by allowing constraints to be part of our rules. We provide a model-theoretic semantics to our language, as well as an operational semantics setting the foundations for the language implementation. Although in this paper we restrict to a particular type of MAS, namely Electronic Institutions [8], we believe that we managed to set the foundations for specify and implement normative systems in general using only our language. This would allow us to implement different models of institutions like [8][21][26].

In section 2 we present the desired properties of an operationalisable normative language. In section 3 we propose the simplest normative language that covers all these requirements. Section 4 summarises the notion of Electronic Institution and details how to capture normative positions of its participating agents. A case study is detailed in section 5. In section 6 we show how to employ our language to capture the expressiveness of other contemporary approaches. Finally, we draw some conclusions and outline future work in section 7.

2 Desiderata for Norm-Oriented MASs

Our main goal is to produce a language that supports the specification of coordination mechanisms in multi-agent systems by means of norms. For this purpose, we identify below the desirable features we expect from our language.

We take the stance that we cannot refer to agents' mentalistic notions, but only to their observable actions. As a result of agents' social interactions, their *normative positions* [21] – permissions, prohibitions and obligations – change. Hence, the first requirement of our language is to support the *explicit management* of agents' normative positions.

Turning our attention to theoretical models of norms, we notice that there is a plethora of deontic logics with different axioms to establish relationships among deontic notions. Thus, we require that our language captures different deontic notions along with their relationships. In other words, the language must be of *general purpose* so that it helps MAS designers to encode any axiomatisation, and thus specify the widest range of normative systems as possible.

In a sense, we pursue a "machine language" for norms on top of which higher-level languages may eventually be accomodated. Along this direction, and from a language designer's point of view, it is fundamental to identify the *norm patterns* (e.g. conditional obligation, time-based permissions and prohibitions, continuous obligation, and so on) in the literature to ensure that our language supports their encoding (as demonstrated in section 6). In this way, not only shall we be guaranteeing the expressiveness of our language, but also addressing pragmatic concerns by providing *design patterns* to guide and ease MAS design.

In order to ease MAS programming, we shall also require our language to be *declarative*, with an implicit execution mechanism to reduce the number of issues designers

ought to concentrate on. As an additional benefit, we expect its declarative nature to facilitate verification of properties of the specifications.

3 A Language for Managing Norms

The building blocks of our language are first-order terms (denoted as T) and atomic formulae (denoted as A). We shall make use of numbers and arithmetic functions to build terms; arithmetic functions may appear infix, following their usual conventions. We adopt Prolog's convention [2] using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants. We also employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation. Atomic formulae with arithmetic relations represent *constraints* on their variables and have a special status, as we explain below. We give a definition of our constraints, a specific subset of our atomic formulae:

Def. 1. *A constraint* C *is any construct of the form* $\mathsf{T} \diamond \mathsf{T}, \diamond \in \{=, \neq, >, \geq, <, \leq\}$

We need to differentiate ordinary atomic formula from constraints. We shall use A' to denote atomic formulae that are *not* constraints.

We now introduce our rules. These are constructs of the form $LHS \rightsquigarrow RHS$, where LHS contains a representation of parts of the current state of affairs which, if they hold, will cause the rule to be triggered. RHS depicts the updates to the current state of affairs, yielding the next state of affairs:

Def. 2. *A rule, denoted as* R*, is defined as:*

$$\begin{aligned} \mathsf{R} &::= LHS \rightsquigarrow RHS \\ LHS &::= \mathsf{As} \wedge LHS \mid \neg\mathsf{As} \wedge LHS \mid \mathsf{As} \mid \neg\mathsf{As} \\ RHS &::= \mathsf{U} \wedge RHS \mid \mathsf{U} \\ \mathsf{As} &::= \mathsf{As} \wedge \mathsf{As} \mid \mathsf{A} \\ \mathsf{U} &::= \oplus\mathsf{A}' \mid \ominus\mathsf{A}' \mid \oplus\mathsf{C} \end{aligned}$$

Some examples of our rules are shown in section 5. The Us (updates) add and remove A's but constraints C can only be added – our rules work by *refining* a state of affairs, always adding constraints. Finally, we make use of a special kind of term, called a *set constructor*, represented $\{\mathsf{A}' \mid LHS\}$. It is useful when we need to refer to all A's in the state of affairs for which LHS holds. For instance,

$$\{utt(auction, W, \mathsf{I}) \mid \mathsf{I} = p(Ag_1, buyer, Ag_2, R, M, T) \wedge T > 20\}$$

stands for the set of utterances in scene $auction$ by buyer agent Ag_1 after time-stamp 20.

The semantics of our rules are defined in terms of state of affairs: rules map an existing state of affairs to a new state of affairs. Intuitively, a state of affairs stores the current state of the environment as a set of atomic formulae:

Def. 3. *A state of affairs* $\Delta = \{\mathsf{A}_0, \ldots, \mathsf{A}_n\}$ *is a a finite and possibly empty set of atomic formulae* $\mathsf{A}_i, 0 \leq i \leq n$.

We now define the semantics of our rules as relationships between states of affairs. We adopt the usual semantics of production rules [14, 20], that is, we exhaustively apply each rule by matching its LHS against the current state of affairs and use the values of variables obtained in this match to instantiate the RHS.

Def. 4. $\mathbf{s}^*(\Delta, LHS \rightsquigarrow RHS, \Delta')$ *holds iff* $\mathbf{s}_l^*(\Delta, LHS, \{\sigma_1, \ldots, \sigma_n\})$ *and* $\mathbf{s}_r(\Delta, RHS \cdot \sigma_i, \Delta'), 1 \leq i \leq n$*, hold.*

That is, two states of affairs Δ and Δ' are related by a rule $LHS \rightsquigarrow RHS$ if, and only if, we obtain all different substitutions $\{\sigma_1, \ldots, \sigma_n\}$ that make the left-hand side match Δ and apply these substitutions to RHS in order to build Δ'.

We now state when the left-hand side of a rule matches a state of affairs. We employ relationship $\mathbf{s}_l(\Delta, LHS, \sigma)$ defined below.

Def. 5. $\mathbf{s}_l(\Delta, LHS, \sigma)$ *holds between state of affairs* Δ*, the left-hand side of a rule* LHS *and a substitution* σ *depending on the format of* LHS*:*

1. $\mathbf{s}_l(\Delta, Atfs' \wedge Constrs, \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, Atfs', \sigma)$ *and* $\mathbf{s}_l(\Delta, Constrs, \sigma)$ *hold.*
2. $\mathbf{s}_l(\Delta, Atf' \wedge Atfs', \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, Atf', \sigma')$ *and* $\mathbf{s}_l(\Delta, Atfs', \sigma'')$ *hold and* $\sigma = \sigma' \cup \sigma''$*.*
3. $\mathbf{s}_l(\Delta, \neg\, Atf', \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, Atf', \sigma)$ *does not hold.*
4. $\mathbf{s}_l(\Delta, Atf', \sigma)$ *holds iff* $Atf' \cdot \sigma \in \Delta$*.*
5. $\mathbf{s}_l(\Delta, (Constr_1 \wedge \cdots \wedge Constr_n), \sigma)$ *holds iff* $constrs(\Delta, C)$ *and* $\{Constr_1 \cdot \sigma, \ldots, Constr_n \cdot \sigma\} \sqsubseteq C$*.*

Case 1 breaks the left-hand side of a rule into its atomic formulae and constraints and defines how their semantics are combined via σ. Cases 2-4 depict the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction. Case 5 formalises the semantics of our constraints when they appear on the left-hand side of a rule: we apply the substitution σ to them (thus reflecting any values of variables given by the matchings of atomic formula), then compare constraints using $\sqsubseteq$.

We want our rules to be *exhaustively* applied on the state of affairs. We thus need relationship $\mathbf{s}_l^*(\Delta, LHS, \Sigma)$ which uses $\mathbf{s}_l$ above to obtain in $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$ *all* possible matches of the left-hand side of a rule:

Def. 6. $\mathbf{s}_l^*(\Delta, LHS, \Sigma)$ *holds, iff* $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *is the largest non-empty set such that* $\mathbf{s}_l(\Delta, LHS, \sigma_i), 1 \leq i \leq n$*, holds.*

We can define the application of a set of a substitutions $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ to a term $Term$: this results in a set of substituted terms, $Term \cdot \{\sigma_1, \ldots, \sigma_n\} = \{Term \cdot \sigma_1, \ldots, Term \cdot \sigma_n\}$. We now define the semantics of the RHS of a rule:

Def. 7. *Relation* $\mathbf{s}_r(\Delta, RHS, \Delta')$ *mapping a state of affairs* Δ*, the right-hand side of a rule* RHS *and another state of affairs* Δ' *is defined as:*

1. $\mathbf{s}_r(\Delta, (Update \wedge RHS), \Delta')$ *holds iff both* $\mathbf{s}_r(\Delta, Update, \Delta_1)$ *and* $\mathbf{s}_r(\Delta, RHS, \Delta_2)$ *hold and* $\Delta' = \Delta_1 \cup \Delta_2$*.*
2. $\mathbf{s}_r(\Delta, \oplus Atf', \Delta')$ *holds iff* $\Delta' = \Delta \cup \{Atf'\}$*.*
3. $\mathbf{s}_r(\Delta, \ominus Atf', \Delta')$ *holds iff* $\Delta' = \Delta \setminus \{Atf'\}$*.*
4. $\mathbf{s}_r(\Delta, \oplus Constr, \Delta') = \mathbf{true}$ *iff* $constrs(\Delta, C)$ *and* $satisfy(C \cup \{Constr\}, C')$ *hold and* $\Delta' = \Delta \cup Constr$*.*

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Cases 2 and 3 cater for the insertion and removal of atomic formulae Atf' which do not conform to the syntax of constraints. Case 4 defines how a constraint is added to a state of affairs Δ: the new constraint is checked for its satisfaction with constraints $C \subseteq \Delta$ and then added to Δ'. We assume the new constraint is merged into Δ: if there is another constraint that subsumes it, then the new constraint is discarded. For instance, if $X > 20$ belongs to Δ, then attempting to add $X > 15$ will yield the same Δ.

In the usual semantics of rules of production systems, the values assigned to those variables in the left-hand side must be passed on to the right-hand side. We capture this by associating the right-hand side with a substitution σ obtained when matching the left-hand side against Δ via relation $\mathbf{s}_l$. In the definition 7 above, we have actually $\mathbf{s}_r(\Delta, RHS \cdot \sigma, \Delta')$ – that is, we have a version of the right-hand side with ground variables whose values originate from the matching of the left-hand side to Δ.

The semantics above define an infinite sequence of states $\langle \Delta_0, \Delta_1, \ldots \rangle$ if $\mathbf{s}^*(\Delta_i, \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}, \Delta_{i+1})$, that is, Δ_{i+1} (obtained by applying the rules to Δ_i) is used to obtain Δ_{i+2} and so on. Figure 1 illustrates how this sequence can accommodate the inter-

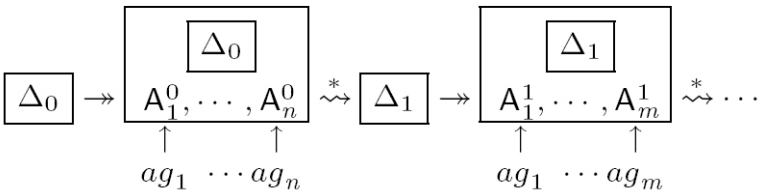

Fig. 1. Semantics as a Sequence of Δ's

vention of agents, that is, their messages. The diagram shows an initial state Δ_0 (possibly empty) that is offered (represented by "$\twoheadrightarrow$") to a set of agents $\{ag_1, \ldots, ag_n\}$. These agents add their utterances $\{\mathsf{A}_1^0, \ldots, \mathsf{A}_n^0\}$ to Δ_0. After the agents add their utterances, then the rules are exhaustively applied (represented by "$\overset{*}{\rightsquigarrow}$") to $\Delta_0 \cup \{\mathsf{A}_1^0, \ldots, \mathsf{A}_n^0\}$. The resulting state Δ_1 is, on its turn, offered to agents, and so on. A meta-interpreter [11] operationalises the semantics above.

4 Electronic Institutions

Our work fits within the context of *electronic institutions* (EIs) [8], providing them with an explicit normative layer. There are two major features in EIs – the *states* and *illocutions* (*i.e.*, messages) uttered (*i.e.*, sent) by those agents taking part in the EI. The states are connected via edges labelled with the illocutions that ought to be sent at that particular point in the EI. Another important feature in EIs are the agents' *roles*: these are labels that allow agents with the same role to be treated collectively thus helping

engineers abstract away from individuals. We define below the class of illocutions we aim at – these are a special kind of term:

Def. 8. *Illocutions* $\bar{\mathsf{i}}$ *are terms* $p(ag, r, ag', r', \mathsf{T}, t)$ *where* p *is an illocutionary particle (*e.g.*,* inform*,*ask*); *ag, ag' *are agent identifiers;* r, r' *are role labels;* T *is a term with the actual content of the message;* $t \in \mathbb{N}$ *is a time stamp.*

We shall refer to illocutions that may have uninstantiated (free) variables within themselves as *illocution schemes* and will be denoted by I.

Another important concept in EIs we employ here is that of a *scene*. Scenes offer means to break down larger protocols into smaller ones with specific purposes. We can uniquely refer to the point of the protocol where an illocution $\bar{\mathsf{i}}$ was uttered by the pair (s, w) where s is a scene name and w is the state from which an edge labelled with I leads to another state.

An institutional state is a state of affairs that stores all utterances during the execution of a MAS, also keeping a record of all obligations, permissions and prohibitions associated with the agents. We differentiate five kinds of atomic formulae in our institutional states Δ, with the following intuitive meanings:

1. ground formulae $oav(o, a, v)$ – object o has an attribute called a with value v.
2. ground formulae $att(s, w, \bar{\mathbf{i}})$ – an agent attempted to utter $\bar{\mathbf{i}}$ at state w of scene s.
3. ground formulae $utt(s, w, \bar{\mathbf{i}})$ – $\bar{\mathbf{i}}$ was uttered at w of s.
4. $obl(s, w, \mathbf{i})$ – $\mathbf{i}$ is *obliged* to be uttered at w of s.
5. $per(s, w, \mathbf{i})$ – $\mathbf{i}$ is *permitted* to be uttered at w of s.
6. $prh(s, w, \mathbf{i})$ – $\mathbf{i}$ is *prohibited* at w of s.
7. $ctr(s, w, st, t_s)$ – s entered to w at state st and time t_s

We shall use formulae 4–6 above to represent normative aspects of agent societies. We only allow fully ground illocutions and attributes (cases 1–3 above) to be uttered[1]; however, in formulae 4-6 s and w may be variables and $\mathbf{i}$ may contain variables. We shall use formulae 7 to represent the state change and time passing. We do not "hardwire"deontic notions in our semantics: the predicates above represent deontic operators but not their relationships. These are captured with rules (also called in this context institutional rules), conferring generality on our approach as different deontic relationships can be forged, as we show below.

5 Example: The Dutch Auction Protocol

In this section, we try to illustrate the pragmatics of our norm-oriented language by specifying the auction protocol employed in the fish market described in [19]. Following

[1] We differentiate between attempts to utter (att) and actual utterances (utt). Since we aim at heterogeneous agents whose behaviour we cannot guarantee, we create a "sandbox" where agents can utter whatever they want and we represent these utterances via att formulae. However, not everything agents say may be in accordance with the protocol specification – the utterances that are not part of the protocol may be discarded or may cause sanctions, depending on the deontic notions we want or need to implement. The utt formulae are thus *confirmations* of the att formulae.

[19], the fish market can be described as a place where several scenes [8] take place simultaneously, at different locations, but with some causal continuity. The principal scene is the auction itself, in which buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order, the so-called *downward bidding protocol*, a variation of the traditional Dutch auction protocol that proceeds as follows:

[Step 1] The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter.
[Step 2] With a chosen good g, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, (p_α) previously fixed by a sellers' admitter, as long as these price quotations are above a *reserve price* (p_{rsv}) previously defined by the seller.
[Step 3] For each price the auctioneer calls, several situations might arise during the open round described below.
[Step 4] The first three steps repeat until there are no more goods left.

The situations arising in step 3 are:

Multiple bids – Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price;
One bid – Only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Otherwise, the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is fined;
No bids – No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price obtained by decreasing the current price according to the price step. Otherwise, the auctioneer declares the good as *withdrawn* and closes the round.

Proposed Solution: These features are captured in figure 2 (for formatting reasons, we will use A_i and As_i to denote, respectively, an atomic formula and a conjunction of them):

Multiple bids – Expression 1 shows a rule that obliges the auctioneer to inform the buyers, whenever a collision comes about, about the collision and to restart the bidding round at a higher price (in this case, 120% the collision price). Notice that X will hold all the utterances at scene $DutchA$ and state $w4$ issued by buyer agents that bid for an item IT at price P at time T_0 after the last offer. We obtain the last offers by checking that there are not further offers whose time-stamp are greater than the time-stamp of the first one. If the number of illocutions in X is greater than one, the rule fires the obligation above;
Timeout – Each bidding round has a associated bidding time that can be regarded as a timeout. Expression 2 shows a rule that checks whether the bidding time has expired since scene $DutchA$ reached state $w4$. Intuitively it means that if we are at state w_4, the amount of time before the timeout is stored by an oav and it is T_o, the last time the scene entered at state w_3 was T and the first time the scene entered at state w_4 after T was T_3 (and time-stamp $Timestamp_3$), D is the current

$$(X = \{ \mathsf{A}_0 | \mathsf{A}_1 \wedge \neg (\mathsf{A}_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge | X | > 1) \rightsquigarrow (\oplus \mathsf{A}_3 \wedge \oplus \mathsf{A}_4 \wedge \oplus (P_2 > P * 1.2))$$

$$\begin{array}{l} \mathsf{A}_0 = utt(DutchA, w4, inform(A_1, buyer, Au, auctioneer, bid(IT, P), T_0)) \\ \mathsf{A}_1 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_1)), \\ where\ \mathsf{A}_2 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_3 = obl(DutchA, w_5, inform(Au, auctioneer, All, buyer, collision(IT, P), T_2)) \\ \mathsf{A}_4 = obl(DutchA, w_3, inform(Au, auctioneer, All, buyer, offer(IT, P_2), T_3)) \end{array} \quad (1)$$

$$\begin{array}{l} \left(\begin{array}{c} ctr(DutchA, w_4, T_0, Ts_0) \wedge now(T_0) \wedge oav(DutchA\_w_4, timeout, T_{out}) \wedge \\ \mathsf{As}_0 \wedge \mathsf{As}_1 \wedge T_3 > T \wedge \mathsf{As}_2 \wedge current\_date(D) \wedge D - Timestamp_3 > T_{out} \end{array} \right) \\ \rightsquigarrow \\ \left(\begin{array}{c} \oplus ctr(DutchA, w_5, T_6, D) \wedge \ominus now(T_0) \wedge \oplus now(T_6) \wedge \\ \oplus (T_6 = T_0 + 1) \wedge \oplus timeout(DutchA, w_4, w_5, T_6) \end{array} \right) \\ \begin{array}{l} \mathsf{As}_0 = ctr(DutchA, w_3, T, Timestamp) \wedge \neg (ctr(DutchA, w_3, T_2, Timestamp_2) \wedge T_2 > T) \\ where\ \mathsf{As}_1 = ctr(DutchA, w_4, T_3, Timestamp_3) \wedge \neg (ctr(DutchA, w_4, T_4, Timestamp_4) \wedge T_4 < T_3) \\ \mathsf{As}_2 = \neg (utt(DutchA, w4, inform(A_1, buyer, Au, auctioneer, bid(IT, P), T_5)) \wedge T5 > T) \end{array} \end{array}$$

(2)

$$\begin{array}{l} (X = \{ \mathsf{A}_0 | \mathsf{A}_1 \wedge \neg (\mathsf{A}_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge | X | = 1 \wedge oav(A_1, credit, C) \wedge C \geq P) \\ \rightsquigarrow \\ (\oplus \mathsf{A}_3) \end{array}$$

$$\begin{array}{l} \mathsf{A}_0 = utt(DutchA, w4, inform(A_1, buyer, Au, auctioneer, bid(IT, P), T_0)) \\ where\ \mathsf{A}_1 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_1)), \\ \mathsf{A}_2 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_3 = obl(DutchA, w_5, inform(Au, auctioneer, All, buyer, sold(IT, P, A_1), T_4)) \end{array} \quad (3)$$

$$(\mathsf{A}_0 \wedge \neg (\mathsf{A}_1 \wedge T2 > T) \wedge oav(Ag, credit, C) \wedge C < P) \rightsquigarrow (\oplus \mathsf{A}_2)$$

$$\begin{array}{l} \mathsf{A}_0 = utt(DutchA, w_3, inform(Au, auctioneer, A, buyer, offer(IT, P), T)) \\ where\ \mathsf{A}_1 = utt(DutchA, w_3, inform(Au, auctioneer, A, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_2 = prh(DutchA, w_4, inform(A, buyer, Au, auctioneer, bid(IT, P_2), T_3)) \end{array} \quad (4)$$

$$\begin{array}{l} \left(\begin{array}{c} X = \{ \mathsf{A}_0 | \mathsf{A}_1 \wedge \neg (\mathsf{A}_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge \\ | X | = 1 \wedge oav(A_1, credit, C) \wedge C < P \end{array} \right) \rightsquigarrow \left(\begin{array}{c} \ominus oav(A_1, credit, C) \wedge \oplus oav(A_1, credit, C_2) \wedge \oplus \mathsf{A}_3 \\ \oplus (C_2 = C - P * 0.1) \wedge \oplus (P_2 = P * 1.2) \end{array} \right) \\ \begin{array}{l} \mathsf{A}_0 = utt(DutchA, w4, inform(A_1, buyer, Au, auctioneer, bid(IT, P), T_0)) \\ where\ \mathsf{A}_1 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_1)), \\ \mathsf{A}_2 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_3 = obl(DutchA, w5, offer(Au, auctioneer, All, buyer, offer(IT, P_2), T_3)) \end{array} \end{array} \quad (5)$$

$$\begin{array}{l} \left(\begin{array}{c} ctr(DutchA, w5, T_n, T_s) \wedge now(T_n) \wedge \mathsf{A}_0 \wedge \neg (\mathsf{A}_1 \wedge T_2 > T) \wedge \\ timeout(DutchA, w_4, w_5, T_3) \wedge T_3 > T \wedge oav(IT, reservation\_price, RP) \wedge \\ oav(IT, decrement\_rate, DR) \wedge RP < P - DR \end{array} \right) \\ \rightsquigarrow \\ (\oplus \mathsf{A}_2 \wedge \oplus (P2 = P - DR)) \end{array}$$

$$\begin{array}{l} \mathsf{A}_0 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T)) \\ where\ \mathsf{A}_1 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_2 = obl(DutchA, w5, offer(Au, auctioneer, All, buyer, offer(IT, P2), T_4)) \end{array} \quad (6)$$

$$\begin{array}{l} \left(\begin{array}{c} ctr(DutchA, w5, T_n, T_s) \wedge now(T_n) \wedge \mathsf{A}_0 \wedge \neg (\mathsf{A}_1 \wedge T_2 > T) \wedge \\ timeout(DutchA, w_4, w_5, T_3) \wedge T_3 > T \wedge oav(IT, reservation\_price, RP) \wedge \\ oav(IT, decrement\_rate, DR) \wedge RP \geq P - DR \end{array} \right) \rightsquigarrow (\oplus \mathsf{A}_2) \\ \begin{array}{l} \mathsf{A}_0 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T)) \\ where\ \mathsf{A}_1 = utt(DutchA, w3, offer(Au, auctioneer, All, buyer, offer(IT, P), T_2)) \\ \mathsf{A}_2 = obl(DutchA, w_5, inform(Au, auctioneer, All, buyer, withdrawn(IT), T_3)) \end{array} \end{array} \quad (7)$$

Fig. 2. Institutional rules for the Dutch auction protocol

date and the time that have passed from $Timestamp_3$ until now is greater than the timeout T_o then we update the current state from $w4$ to $w5$, change the current time ($now(T_6)$) and register the timeout in the model. We use the predicate ctr to store the history of states ordered by a natural T_0 and time-stamped Ts_0

One bid - Winner determination – Expression 3 states that if only one bid has occurred during the current bidding round and the credit of the bidding agent is greater or equal than the price of the good in auction, the rule adds the obligation for the auctioneer to inform all the buyers about the sale;

One bid - Unsupported bid – In the current formalization of electronic institutions [8], illegal illocutions are prevented by rejecting them. Therefore, agents can not be punished when acting illegaly.

a) Prevention – Expression 4 shows a rule that prevents agents to issue bids whose credit can notpay for. It states that if P is the last offer called by an auctioneer for item IT, at scene $DutchA$ and state w_3, and agent Ag's credit is less than P, then agent Ag is prohibited to bid;

b) Punishment – In case we do not want to reject illegal actions, we must provide methods for discouraging agents to violate norms. For instance, by applying punishments when norms are violated. Expression 5 shows a rule that punishes an agent when issuing a winning bid he can not pay for. More precisely, the rule punishes an agent (A_1) by decreasing his credit 10% the value of the good at auction. The oav predicate on the LHS of the rule is used for gathering the value of the current credit of the offender agent. The rule also add an obligation for the auctioneer to restart the bidding round and the constraint that the new offer should be greater than 120% the old price;

No bids - New Price – Expression 6 shows a rule that checks if there were no bids and the next price is greater than the reservation price. If so, it adds the obligation for the auctioneer to start a new bidding round. Rule 6 checks that the current scene state is $w5$, whether a timeout has expired after the last offer and whether the new price is greater than reservation price. If so, the rule adds the obligation for the auctioneer to offer the item at a lower price. By retrieving the last offer we gather the last offer price. By checking the oav predicates we gather the values of the reservation price and the decrement rate for item IT. 5)

No bids - Withdrawal – Expression 7 shows a rule that checks if there were no bids and the next price is less than the reservation price, then adds the obligation for the auctioneer to withdraw the item. Rule 7 checks that the current institutional state is $w5$, whether a timeout has occurred after the last offer and whether the new offer price is greater than reservation price. If the LHS holds, the rule fires to add the obligation for the auctioneer to withdraw the item. By checking the last offer we gather the last offer price. By checking the oav predicates we gather the values of the reservation price and the decrement rate for item IT.

6 Expressiveness Analysis

In this section we analyse the expressiveness of our language by comparing it with other recent approaches.

6.1 Conditional Deontic Logic with Deadlines

As shows the BNF definition of figure 3 [24, 25], a norm description is composed by several fields.

The norm condition is the declarative norms, as obtained from, for instance, the legal domain. The other fields in the norm description are; 1) the *violation condition* which is a formula defining when the norm is violated, 2) the *detection mechanism* which describes the mechanisms included in the agent platform that can be used for detecting violations, 3) the *sanctions* which defines the actions that are used to punish the agent(s) violation of the norm, and 4) the *repairs* which is a set of actions that are used for recovering the system after the occurrence of a violation.

$$
\begin{aligned}
\text{NORM} &::= \text{NORM\_CONDITION} \\
&\quad\ \ \text{VIOLATION\_CONDITION} \\
&\quad\ \ \text{DETECTION\_MECHANISM} \\
&\quad\ \ \text{SANCTIONS} \\
&\quad\ \ \text{REPAIRS} \\
\text{VIOLATION\_CONDITION} &::= formula \\
\text{DETECTION\_MECHANISM} &::= \{action\ expressions\} \\
\text{SANCTIONS} &::= \text{PLAN} \\
\text{REPAIRS} &::= \text{PLAN} \\
\text{PLAN} &::= action\ expression \mid action\ expression\ ;\ \text{PLAN}
\end{aligned}
$$

Fig. 3. BNF of Utrecht's norms

As the definition of figure 4 shows, norms can be deontic notions as either permissions, obligations or prohibitions. Furthermore, norms can be related to actions or to predicates (states). The former restrict or allow the actions that a set of agents can perform, the latter constrain the results of the actions that a set of agents can perform. This results are predicates that can be done true or false. It is forbidden that tom performs the action of smoke ($\mathsf{FORBIDDEN}\ (tom \mathsf{DO} smoke)$) and it is forbidden that tom brings about that the air is polluted ($\mathsf{FORBIDDEN}\ (tom, polluted(air))$) are two examples of the types of norm stated above.

$$
\begin{aligned}
\text{NORM\_CONDITION} &::= N(S\ \langle \mathsf{IF}\ C\rangle) \mid \mathsf{OBLIGED}(a\ \mathsf{ENFORCE}(N(a, S\langle \mathsf{IF}\ C\rangle))) \\
N &::= \mathsf{OBLIGED} \mid \mathsf{PERMITTED} \mid \mathsf{FORBIDDEN} \\
J &::= (a, P) \mid (a \mathsf{DO} A) \\
S &::= J \mid J\ \mathsf{TIME} \mid J\ \mathsf{ACTION} \\
C &::= formula \\
P &::= predicate \\
A &::= action\ expression \\
\text{TIME} &::= \mathsf{BEFORE} D \mid \mathsf{AFTER} D \\
\text{ACTION} &::= \mathsf{BEFORE} J \mid \mathsf{AFTER} J
\end{aligned}
$$

Fig. 4. BNF of the norm condition

Through the condition (C) and temporal operators (BEFORE and AFTER), norms can be made only applicable to certain situations. Conditions and temporal operators are considered optional. Temporal operators can be applied to a deadline (D) or to an action or predicate (J).

Now we are going to expose the translation of the norms presented above into our rule language. Since we consider illocutions as the only actions that can be performed in a electronic institution, actions need to be translated into illocutions uttering that the action has been done.

Utrecht norms	Institutional rules
PERMITTED$((A\text{DO}utter(S, W, I)))$	$scene(S, W) \wedge att(S, W, I)$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN$((A\text{DO}utter(S, W, I)))$	$scene(S, W) \wedge att(S, W, I)$ $\Rightarrow \ominus att(S, W, I) \oplus vio$
OBLIGED$((A\text{DO}utter(S, W, I)))$	a) $scene(S, W) \wedge \neg\ utt(S, W, I) \Rightarrow \oplus vio$ b) $scene(S, W) \wedge obl(S, W, I) \wedge$ $utt(S, W, I) \Rightarrow \ominus obl(S, W, I)$

Table 1. Translation of general norms into rules

Table 1 shows the translation of general norms into our rules. The permission of an action can be translated into a rule the converts the attempt to utter the I illocution at state W of scene S ($att(S, W, I)$) into the result of the illocution being uttered ($utt(S, W, I)$). The prohibition of an action can be translated into a rule that ignores the attempt to utter the illocution, and optionally can sanction the violation ($vio(id)$). For space reasons, we use the predicate vio to represent that the proper sanctions and repairs are executed. We assume that there exists another rule with the vio predicate on the LHS and sanctions and repairs on the RHS. The obligation of an action needs to be translated into two rules. The former rule sanctions the obliged agents if they do not utter the expected illocution in the scene and state in which they are obliged to do it. The latter rule removes the obligation once the obliged action has been done.

Utrecht norms	Institutional rules
PERMITTED$((A\text{DO}utter(S, W, I))$**if** $C)$	$scene(S, W) \wedge C \wedge att(S, W, I)$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN$((A\text{DO}utter(S, W, I))$**if** $C)$	$scene(S, W) \wedge C \wedge att(S, W, I)$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus vio(1)$
OBLIGED$((A\text{DO}utter(S, W, I))$**if** $C)$	a) $scene(S, W) \wedge C \wedge \neg\ utt(S, W, I) \Rightarrow \oplus vio(2)$ b) $scene(S, W) \wedge C \wedge obl(S, W, I) \wedge$ $utt(S, W, I) \Rightarrow \ominus obl(S, W, I)$

Table 2. Translation of conditional norms into rules

Table 2 shows the translation of conditional norms into our rules. This translation can be done in a similar way to the translation done in the previous table but adding a condition (C) on the LHS of the rule.

Utrecht norms	Institutional rules
PERMITTED$((A$DO$utter(S, W, I))$BEFORE$D)$	$scene(S, W) \wedge att(S, W, I) \wedge T < D$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN$((A$DO$utter(S, W, I))$BEFORE$D)$	$scene(S, W) \wedge att(S, W, I) \wedge T < D$ $\Rightarrow \ominus att(S, W, I) \oplus vio(3)$
OBLIGED$((A$DO$utter(S, W, I))$BEFORE$D)$	a) $scene(S, W) \wedge \neg\, utt(S, W, I) \wedge$ $current\_time(T_1) \wedge T_1 > D \Rightarrow \oplus vio(4)$ b) $scene(S, W) \wedge utt(S, W, I) \wedge T > D \Rightarrow \oplus vio(5)$ c) $scene(S, W) \wedge obl(S, W, I) \wedge$ $utt(S, W, I) \wedge T < D \Rightarrow \ominus obl(S, W, I)$

Table 3. Translation of "BEFORE $time$" norms into rules

Table 3 shows the translation of norms with the BEFORE $time$ construct into our rules. This translation can be done likewise the translation done in the table 1 but adding in the LHS of the rule the condition that the time in which the attempt is done (T) has to be less that the deadline (D). Now in the translation of obligations we need three rules: one to sanction the agents that do not utter the expected illocution before the deadline, other to sanction the agents that utter the expected illocution late and another rule to remove the obligation if the illocution is uttered before the deadline.

Utrecht norms	Institutional rules
PERMITTED$((A$DO$utter(S, W, I))$AFTER$D)$	$scene(S, W) \wedge att(S, W, I) \wedge T > D$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN$((A$DO$utter(S, W, I))$AFTER$D)$	$scene(S, W) \wedge att(S, W, I) \wedge T > D$ $\Rightarrow \ominus att(S, W, I) \oplus vio(6)$
OBLIGED$((A$DO$utter(S, W, I))$AFTER$D)$	$scene(S, W) \wedge obl(S, W, I) \wedge utt(S, W, I) \wedge$ $T > D \Rightarrow \ominus obl(S, W, I)$

Table 4. Translation of "AFTER $time$" norms into rules

Table 4 shows the translation of norms with the construct AFTER $time$ into our rules. This translation can be done in a similar way to the translation done in the table 1 but adding to the LHS of the rule the condition that the time in which the attempt is done (T) has to be greater that the deadline (D). Now in the translation of obligations we only need one rule to remove the obligation if the illocution is uttered after the specified time. In the current implementation of electronic institutions obligations must be satisfied the first time the agents are in the expected scene and state. However, as we do not assume that, this norm can not be sanctioned.

Table 5 shows the translation of norms with the construct BEFORE $action$ into our rules. The translation of a permission before another utterance is done by adding two rules that transform an attempt to utter into the illocution being uttered, first if the second action has not been performed yet or if the second illocution has been uttered but after the permitted illocution. The translation of a prohibition before another utterance is also done by adding two rules similar to the rules for the permitted before case but changing the RHS to sanction the violation. The translation of an obligation before another utterance is done by means of four rules; a) sanctions when the permitted illocution has not been uttered and the deadline illocution has been uttered; b) sanc-

Utrecht norms	Institutional rules
PERMITTED($(A$DO$utter(S, W, I))$) BEFORE(BDO$utter(S_2, W_2, I_2)$))	a) $scene(S, W) \wedge att(S, W, I) \wedge \neg\ utt(S_2, W_2, I_2)$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$ b) $scene(S, W) \wedge att(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge T < T_2$ $\Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN($(A$DO$utter(S, W, I))$) BEFORE(BDO$utter(S_2, W_2, I_2)$))	a) $scene(S, W) \wedge att(S, W, I) \wedge \neg\ utt(S_2, W_2, I_2)$ $\Rightarrow \ominus att(S, W, I) \oplus vio(7)$ b) $scene(S, W) \wedge att(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge T < T_2$ $\Rightarrow \ominus att(S, W, I) \oplus vio(8)$
OBLIGED($(A$DO$utter(S, W, I))$) BEFORE(BDO$utter(S_2, W_2, I_2)$))	a) $scene(S, W) \wedge \neg\ utt(S, W, I) \wedge utt(S_2, W_2, I_2)$ $\Rightarrow \oplus vio(9)$ b) $scene(S, W) \wedge utt(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge$ $T > T_2 \Rightarrow \oplus vio(10)$ c) $scene(S, W) \wedge obl(S, W, I) \wedge utt(S, W, I) \wedge$ $\neg\ utt(S_2, W_2, I_2) \Rightarrow \ominus obl(S, W, I)$ d) $scene(S, W) \wedge obl(S, W, I) \wedge utt(S, W, I) \wedge$ $utt(S_2, W_2, I_2) \wedge T < T_2 \Rightarrow \ominus obl(S, W, I)$

Table 5. Translation of "BEFORE $action$" norms into rules

tions when the deadline illocution has been uttered before the permitted illocution; c) removes the obligation if it is fulfilled and the deadline illocution has not been uttered; and d) removes the obligation if it is fulfilled before the deadline illocution has been uttered.

Utrecht norms	Institutional rules
PERMITTED($(A$DO$utter(S, W, I))$) AFTER(BDO$utter(S_2, W_2, I_2)$))	$scene(S, W) \wedge att(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge$ $T > T_2 \Rightarrow \ominus att(S, W, I) \wedge \oplus utt(S, W, I)$
FORBIDDEN($(A$DO$utter(S, W, I))$) AFTER(BDO$utter(S_2, W_2, I_2)$))	$scene(S, W) \wedge att(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge$ $T > T_2 \Rightarrow \ominus att(S, W, I) \oplus vio(11)$
OBLIGED($(A$DO$utter(S, W, I))$) AFTER(BDO$utter(S_2, W_2, I_2)$))	$scene(S, W) \wedge obl(S, W, I) \wedge utt(S, W, I) \wedge utt(S_2, W_2, I_2) \wedge$ $T > T_2 \Rightarrow \ominus obl(S, W, I)$

Table 6. Translation of "AFTER $action$" norms into rules

Table 6 shows the translation of norms with the AFTER $action$ construct into our rules. The translation of these type of norms is simply done by checking if the time when the attempt to utter an illocution is greater than the time of the deadline illocution.

We will not deeply cope with the bringing about of predicates. But they can be translated into rules in a similar way as stated above. For that purpose, we use a new predicate to represent an attempt to bring about something: $att(A, R, P, T)$. That means that an agent A who enacts role R attempts to bring about P at time T. We use $brg(A, R, P, T)$ to represent that an agent A who enacts role R brought about P at time T. Table 7 shows the translation of norms with predicates into rules.

Hence, the norms defined by Dignum et al. can be translated into normative rules by adding the violation condition into the LHS of the rule and sanctions and repairs into the RHS as the following rule scheme shows:

$$VC \Rightarrow S \wedge R$$

Utrecht norms	Institutional rules
PERMITTED$((A, P))$	$now(T) \wedge att(A, R, P, T) \Rightarrow \ominus att(A, R, P, T) \wedge \oplus brg(A, R, P, T) \wedge \oplus P$
FORBIDDEN$((A, P))$	$now(T) \wedge att(A, R, P, T) \Rightarrow \ominus att(A, R, P, T) \oplus vio(1)$
OBLIGED$((A, P))$	$now(T) \wedge obl(A, R, P, T_1) \wedge brg(A, R, P, T) \Rightarrow \ominus obl(A, R, P, T_1)$
PERMITTED$((A, P)$**if** $C)$	$now(T) \wedge C \wedge att(A, R, P, T) \Rightarrow \ominus att(A, R, P, T) \wedge \oplus brg(A, R, P, T) \wedge \oplus P$
FORBIDDEN$((A, P)$**if** $C)$	$now(T) \wedge C \wedge att(A, R, P, T) \Rightarrow \ominus att(A, R, P, T) \oplus vio(2)$
OBLIGED$((A, P)$**if** $C)$	$now(T) \wedge C \wedge \neg\, brg(A, R, P, T) \Rightarrow \oplus vio(3)$ $now(T) \wedge C \wedge obl(A, R, P, T_1) \wedge brg(A, R, P, T) \Rightarrow \ominus obl(A, R, P, T_1)$

Table 7. Translation of some "Bring about" norms into rules

where VC is the violation condition, S and R stands respectively for sanctions and repairs, all of them extracted from the norm.

6.2 Z Specification of Norms

Although Luck et al. proposed a framework that covers several topics of a normative multi-agent system, we shall focus on their definition of norm [18]. As figure 5 shows a norm is composed of several fields: $addressees$ stands for the set of agents that have to comply with the norm; $beneficiaries$ stands for the set of agents that profit from the compliance of the norm; $normativegoals$ stands for the set of goals that ought to be achieved by addressee agents; $rewards$ are received by addressee agents if they satisfy the normative goals; $punishments$ are imposed to addressee agent when they do not satisfy the normative goals; $context$ specifies the preconditions to apply the norm and $exceptions$ when it is not applicable. As we see in their definition, a norm always must have addressees, normative goals and context. We can also see that $rewards$ and $punishments$ are disjoint sets, as well as $context$ and $exceptions$ also are.

$$
\begin{array}{|l}
\textit{Norm} \\
\hline
addresses, beneficiaries : \mathbb{P}\, Agent \\
normativegoals, rewards, punishments : \mathbb{P}\, Goal \\
context, exceptions : \mathbb{P}\, EnvState \\
\hline
normativegoals \neq \emptyset;\ addresses \neq \emptyset;\ context \neq \emptyset \\
context \cap exceptions = \emptyset;\ rewards \cap punishments = \emptyset \\
\hline
\end{array}
$$

Fig. 5. Z definition of a Luck's norm

The contextualisation of the norms includes linking the addressee, beneficiaries and normative goal to the correct corresponding utterance, as well as defining the predicates used in the eInstitution to express the context and exceptions.

Using the language introduced in section 3 we can again show that norms specified in this norm frame can be operationalised. After contextualisation the norms can be

easily translated into the following normative rule to detect violations of the norm:

$$(context \wedge \sim exception \wedge \neg\ goal') \Rightarrow punishments$$

where $context$ and $exception$ are predicates obtained through the contextualisation for specifying the context and exceptions mentioned in the norm, $goal'$ is the contextualised normative goal (thus including the addressee and possible beneficiaries), and the $\sim$ operator for expressing negation-as-failure (since no exceptions might be given). $punishments$ are contextualised actions obtained from the norm. This rule means that in a particular context that it is not an exception of the norm and the goal has not been fulfilled the actions defined by $punishments$ should be executed. If rewards are specified, the following normative rule can be also obtained:

$$(context \wedge \sim exception \wedge goal') \Rightarrow rewards$$

where $rewards$ are also contextualised actions obtained from the norm. This rule specifies that a reward should be given when $addressee$ agents comply with the norm, which is when the norm is applicable and the contextualised normative goal ($goal'$) has been achieved.

6.3 Event Calculus

In [3], Artikis et al. propose the use of event calculus for the specification of protocols. The event calculus is a formalism to represent reasoning about actions or events and their effects in a logic programming framework. It is based on a many-sorted first-order predicate calculus. Figure 6 shows the main predicates of the event calculus.

Predicate	**Meaning**
happens(Act, T)	Action Act occurs at time T
initially($F = V$)	The value of fluent F is V at time 0
holdsAt($F = V$, T)	The value of fluent F is V at time T
initiates(Act, $F = V$, T)	The occurrence of action Act at time T initiates a period of time for which the value of fluent F is V
terminates(Act, $F = V$, T)	The occurrence of action Act at time T terminates a period of time for which the value of fluent F is V

Fig. 6. Main Predicates of the Event Calculus

Predicates that change with time are called *fluents*. As figure 7 shows, obligations, permissions, empowerments, capabilities and sanctions are formalized by means of the following fluents : $obl(Ag, Act)$, $per(Ag, Act)$, $pow(Ag, Act)$, $can(Ag, Act)$ and $sanction(Ag)$. Prohibitions are not formalized in the example of [3] as a fluent since they assume that every not permitted action is forbidden by default.

Fluent	Domain	Meaning
$requested(S, T)$	boolean	subject S requested the floor at time T
$status$	$\{free, granted(S, T)\}$	the status of the floor: $status = free$ denotes that the floor is free whereas $status = granted(S, T)$ denotes that the floor is granted to subject S until time T
$best\_candidate$	agent identifiers	the best candidate for the floor
$can(Ag, Act)$	boolean	agent Ag is capable of performing Act
$pow(Ag, Act)$	boolean	agent Ag is empowered to perform Act
$per(Ag, Act)$	boolean	agent Ag is permitted to perform Act
$obl(Ag, Act)$	boolean	agent Ag is obliged to perform Act
$sanction(Ag)$	$\mathbb{Z}^*$	the sanctions of agent Ag

Fig. 7. Main Fluents of the Artikis' example

Figure 8 shows an example of obligation specified in Event Calculus extracted from [3]. The obligation that C revokes the floor holds at time T if C enacts the role of chair and the floor is granted to someone else different from the best candidate.

$$\mathsf{holdsAt}(obl(C, revoke\_floor(C)) = \mathsf{true}, T) \leftarrow$$
$$role\_of(C, chair), \mathsf{holdsAt}(status = granted(S, T'), T), (T \geq T'),$$
$$\mathsf{holdsAt}(best\_candidate = S', T), (S \neq S')$$

Fig. 8. Example of Obligation in Event Calculus

If we translate all the $holdsAt$ predicates into $uttered$ predicates, we can translate the obligations and permissions of the example by including the rest of conditions in the LHS of the normative rules. However, since there is no concrete definition of norm, we can not state that Artikis' approach is fully translatable into normative rules.

Although event calculus models time, the deontic fluents specified in the example of [3] are not enough to inform an agent about all types of duties. For instance, to inform an agent that it is obliged to perform an action before a deadline, it is necessary to show the agent the obligation fluent and the part of the theory that models the violation of the deadline.

6.4 Hybrid Metric Interval Temporal Logic

Cranefield proposes in [6] the use of rules written in a modal logic with temporal operators called hyMITL$^{\pm}$. It combines CTL$^{\pm}$ with Metric Interval Temporal Logic (MITL) as well as features of hybrid logics. He use the technique of formula progression from the planning system TLPlan to monitor social expectations until they are fulfilled or violated.

Expression 10 shows an example of rule in hyMITL. This rule states that if the current state is one in which c has just made payment for the service, and the current

state is within the one week period from the time this offer is made (time t) then weekly reports will be sent during the next 52 weeks until p optionally cancels the order.

$$\begin{array}{l}\mathsf{AG}^{+}(Done(c, make\_payment(c, p, amount, prod\_num)) \wedge [t, t + 1\text{week}) \rightarrow \\ \downarrow^{week} w.((\neg\, \mathsf{F}^{-}_{[-0,w]} Done(p, send\_report(c, prod\_num, w)) \rightarrow \\ \qquad \mathsf{F}^{+}_{[+0,w+1\text{week}]} Done(p, send\_report(c, prod\_num, w))) \\ \qquad \mathsf{W}^{+}_{[+0,w+52\text{weeks}]} \\ \qquad Done(c, cancel\_order(c, p, prod\_num)))\end{array} \tag{10}$$

Rule 11 shows the translation of the previous hyMITL$^{\pm}$ rule into our rule language. We calculate the number of weeks since the last utterance of payment and the time in which this week ends. If the number of weeks is less than 52 and the report for that week has not been done then the payed agent is obliged to send the report before the end of the week.

$$\left(\begin{array}{c}\mathsf{A}_0 \wedge \neg\, (\mathsf{A}_1 \wedge T_1 > T_0) \wedge \\ current\_date(T_n) \wedge \\ W = trunc((T_n - T_0)/\mathsf{A}) \wedge \\ W < 52 \wedge \neg\, \mathsf{A}_2 \wedge \\ T_{end\_w} = T_0 + (W + 1) * \mathsf{A}\end{array}\right) \rightsquigarrow \left(\begin{array}{c}\oplus \mathsf{A}_3 \wedge \\ \oplus (T_i < T_{end\_w})\end{array}\right)$$

$$\begin{array}{l}where \\ \\ \mathsf{A} = (6048 * 10^5) \\ \mathsf{A}_0 = utt(payment,\, W_0, inform(C, customer, P, payee, \\ \qquad\qquad pay(Amount, Prod\_num),\, T_0)) \\ \mathsf{A}_1 = utt(report,\, W_1, inform(C, customer, P, payee, \\ \qquad\qquad cancel(Prod\_num),\, T_1)) \\ \mathsf{A}_2 = utt(report,\, W_2, inform(P, payee, C, customer, \\ \qquad\qquad send\_report(R,\, W),\, T_2)) \\ \mathsf{A}_3 = obl(report,\, W_2, inform(P, payee, C, customer, \\ \qquad\qquad send\_report(R,\, W),\, T_3))\end{array} \tag{11}$$

Our rules are equivalent to $\mathsf{AG}^{+}(LHS \rightarrow \mathsf{X}^{+} RHS)$ where LHS and RHS are atomic formulae without any temporal operator. As we build the next state of affairs by applying the operations in the RHS of the fired rules, we can not use any other temporal operator in the RHS of our rules. Furthermore, since our state of affairs is non monotonic we can not reason over the past of any formulae. We can only do it in predicates with time-stamps, like the utt predicate, that are not removed from the state of affairs.

We can capture the meaning of the X^{-} operator when it is used in the LHS of the hyMITL rule:
$\mathsf{X}^{-}\phi \simeq ctr(S,\, W,\, T,\, Ts) \wedge \phi(T_0) \wedge T_0 = T - 1$
We can also translate the U^{+} operator when it is used in the RHS of the hyMITL rule:
$\phi\ \mathsf{U}^{+}\psi \simeq \psi \rightsquigarrow \ominus\phi$
Although we can not use all the temporal operators in the RHS of our rules, we can get equivalent results by imposing certain restrictions in the set of rules. $\mathsf{F}^{+}\phi$ can be achieved if $\oplus\phi$ appears in the RHS of a rule and it is possible that the rule fires. $\mathsf{G}^{+}\phi$ can be achieved after ϕ is added and no rule that could fire removes it. Time intervals can be translated into comparisons of time-points as shown in the previous example.

6.5 Social Integrity Constraints

A language called Social Integrity Constraints (SIC) is proposed in [1] and checks whether some events have occurred and some conditions hold and adds new expectations, optionally with constraints.

$$\mathbf{H}(request(B, A, P, D, T_r)) \wedge \mathbf{H}(accept(B, A, P, D, T_a)) \wedge T_r < T_a \\ \rightarrow \mathbf{E}(do(A, B, P, D, T_d)) : T_d < T_a + \tau$$

The expression above shows an example of SIC that intuitively means: if agent B sent a request P to agent A at time T_r, in the context of dialogue D, and A sent an $accept$ to B's request at a later time T_a, then A is expected to do P before a deadline $T_a + \tau$.

The translation of their SICs is very straightforward, because events (**H**) can be translated into our att predicates. Since we also allow predicates to be restricted by constraints, expectations can be translated directly into obligations.

Although syntactically their language is very similar to ours, they are semantically different. Contrarily to them, that use abduction and CHR to execute their rules, we use a forward chaining approach.

Despite the fact that expectations they use are quite similar to obligations, their approach also lack of the rest of deontic notions we manage (permissions and prohibitions). Furthermore, although they mention how expectations are treated, that is, what happens when a expectation is fulfilled or when it is not, and state the possibility of SICs being violated, they do not offer mechanisms to regulate agent's behaviour like the punishment of offender agents or repairing actions that the system can do.

6.6 Object Constraint Language

In [9], it is proposed the use of Object Constraint Language (OCL) for the specification of artificial institutions.

The expression below shows an example of norm written in OCL. This norm commits the auctioneer not to declare a price lower than the agreed reservation price.

As we saw in section 5 we can express with a easier syntax (rule 7) the case that auctioneer is obliged to withdraw the good when the Dutch auction arrives to a price lower than the reservation price. As for [9], we can not perform an exhaustive analysis of the language because they do not offer nor the syntax specification nor the semantics specification.

```
    within h : AuctionHouse
on e : InstitutionalRelationChange(h.dutchAuction,
            auctioneer, created)
if true then
foreach agent in h.employee →
            select(em | e.involved → contains(em))
      do makePendingComm(agent,
            DutchInstAgent(notSetCurPrice(
            h.dutchAuction.id,
            ?p[?p < h.agreement.reservationPrice]),
            < now, now + time_of(e1 : InstStateChange(
            h.dutchAuction, OpenDA, ClosedDA)) >, ∀))
```

As far as we know, they found their theory on the notion of commitment which is quite similar to obligations. In this sense, they do not have the notions of permissions and prohibitions as we do.

6.7 Standard Production Systems

García-Camino [10] proposes the translation of the normative language presented in [24] into Jess rules [13] to monitor and enforce norms. This language captures the deontic notions of permission, prohibition and obligation in several cases: absolute norms, conditional norms, norms with deadline and norms in temporal relation with another event. Absolute norms are directly translated into JESS facts; conditional norms are directly translated into rules that add the deontic facts when the condition holds; norms with deadline are translated into rules that add conditional norms after the deadline has passed. Finally, norms in temporal relation with other events are translated into rules that check if those events have occurred.

Our proposal also bears strong similarities with the work of Lopes Cardoso and Oliveira [16, 15] where norms are also represented as rules of a production system. We notice that our institutional rules can express their notions of contracts and their monitoring (*i.e.*, fulfillment and violation of obligations). However, in [16] constraints can only be used to depict the right-hand side of a rule, that is, the situation(s) when a rule is applicable – constraints are not manipulated the way we do. Furthermore, in that work there is no indication as to how individual agents will know about their normative situation; a diagram introduces the architecture, but it is not clear who/what will apply the rules to update the normative aspects of the system nor how agents synchronise their activities. It is not clear how their approach can be used to engineer open, heterogeneous MASs.

The advantages of using our language, instead of production systems, to specify and monitor the normative position of the agents conforming a MAS are that we can use constraint solving techniques to handle with predicates restricted with constraints and that we borrow unification from Prolog that permits higher level pattern matching. The latter advantage is very useful when specifying complex data structures as, i.e., illocutions. As they can have atomic formulae as the content of the message, they become a second order formula and you need to make programming tricks to express them as a production system fact. In this sense, the programming of higher order formulae as facts gets inviable.

7 Discussion, Conclusions and Future Work

In the beginning of this article we proposed some features that a normative language should embody. In section 3 we proposed a language to model theory change, which we used in sections 4 and 5 to express the normative positions of a set of agents. The case study of section 5 set out the pragmatics of the language and how it incorporates the features detailed in section 2.

In section 6 we compared our language with other contemporary approaches and we found that a wide range of constructs and notions in those languages can be translated into our rule language by using predicates and constraints. After analysing those

languages, we found many features in common. Some are used to capture the deontic notions of permission, prohibition and obligation in predicates or norm constructs. In some cases these deontic notions can be conditional or can be related in time with other events (deadlines or actions). At least, two of the languages capture the notion of sanction as well as the specification of the actions to perform when a violation of a norm occurs. Similarly, rewards specify the actions to perform when a norm has been fulfilled.

Our future work includes the generalisation of the language to cope with actions. We also need to explore the possibility of storing all the past events to reason over the past and the possibility of adding time operators to reduce the amount of code in the LHS of a rule.

References

1. Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Specification and Verification of Agent Interactions using Integrity Social Constraints. Technical Report DEIS-LIA-006-03, Università degli Studi di Bologna, October 2003.
2. K. R. Apt. *From Logic Programming to Prolog*. Prentice-Hall, U.K., 1997.
3. A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. volume 3476 of *LNCS*. Springer-Verlag.
4. Robert Axelrod. *The complexity of cooperation: agent-based models of competition and collaboration*. Princeton studies in complexity. Princeton University, New Jersey, 1997.
5. G. Boella and L. van der Torre. Permission and Obligations in Hierarchical Normative Systems. In *Procs. 8th Int'l Conf. in AI & Law (ICAIL'03)*, Edinburgh, 2003. ACM.
6. S. Cranefield. A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. Technical Report 2005/01, University of Otago, February 2005.
7. Frank Dignum. Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.
8. M. Esteva. *Electronic Institutions: from Specification to Development*. PhD thesis, Universitat Politècnica de Catalunya (UPC), 2003. IIIA monography Vol. 19.
9. Nicoletta Fornara, Francesco Viganò, and Marco Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In *AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM)*, Utrecht, 2005.
10. A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *4th Int'l Joint Conf on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005. Forthcoming.
11. A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. AUCS/TR0503, Dept of Computing Sc., Univ. of Aberdeen, Aberdeen, UK, 2005.
12. N.R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems*, 1:7–38, 1998.
13. Jess. The Rule Engine for Java. Sandia Nat'l Labs. `http://herzberg.ca.sandia.gov/jess`, October 2005.
14. Bryan Kramer and John Mylopoulos. Knowledge Representation. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1. John Wiley & Sons, 1992.
15. Henrique Lopes Cardoso and Eugénio Oliveira. Towards an Institutional Environment using Norms for Contract Performance. volume In press of *LNAI*. Springer-Verlag, 2005.
16. Henrique Lopes Cardoso and Eugénio Oliveira. Virtual Enterprise Normative Framework within Electronic Institutions. volume In press of *LNAI*. Springer-Verlag, 2005.

17. F. López y López. *Social Power and Norms: Impact on agent behaviour*. PhD thesis, University of Southampton, June 2003.
18. F. López y López and Michael Luck. A Model of Normative Multi-Agent Systems and Dynamic Relationships. volume 2934 of *LNAI*. Springer-Verlag, 2004.
19. P. Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona (UAB), 1997. IIIA monography Vol. 8.
20. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Inc., U.S.A., 2 edition, 2003.
21. M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.
22. Y. Shoham and M. Tennenholtz. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
23. R. Tuomela and M. Bonnevier-Tuomela. Norms and Agreement. *European Journal of Law, Philosophy and Computer Science*, 5:41–46, 1995.
24. Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Implementing Norms in Multiagent Systems. volume 3187 of *LNAI*. Springer-Verlag.
25. Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Norms in Multiagent Systems: Some Implementation Guidelines. In *2nd European Workshop on Multi-Agent Systems*, Barcelona, 2004.
26. M. Verdicchio and M. Colombetti. A logical model of social commitment for agent communication. In *Procs. AAMAS 2003*. ACM Press, 2003.
27. Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, February 2002.