Norm-Oriented Programming of Electronic Institutions: A Rule-based Approach

Andrés García-Camino[1], Juan-Antonio 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, University of Aberdeen,
Aberdeen AB24 3UE, United Kingdom
wvasconcelos@acm.org

Abstract. Norms constitute a powerful coordination mechanism among heterogeneous agents. We propose means to specify and explicitly manage the normative positions of agents (permissions, prohibitions and obligations), with which distinct deontic notions and their relationships can be captured. Our rule-based formalism includes constraints for more expressiveness and precision and allows the norm-oriented programming of electronic institutions: normative aspects are given a precise computational interpretation. Our formalism has been conceived as a machine language to which other higher-level normative languages can be mapped, allowing their execution.

1 Introduction

A major challenge in multi-agent system (MAS) research is the design and implementation of *open* multi-agent systems in which coordination must be achieved among agents defined with different languages by several designers who may not trust each other. Norms can be used for this purpose as a means to regulate the observable behaviour of agents as they interact in pursuit of their goals [1–3]. There is a wealth of socio-philosophical and logic-theoretical literature on the subject of norms (*e.g.*, [4, 5]), and, more recently, much attention is 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 in the design and execution of MASs (e.g. [6–10]).

A normative position [4] 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. Ideally, norms, once captured via some suitable formalism, should be directly executed, thus realising a computational, normative environment wherein agents interact. This is what we mean by *norm-oriented programming.* We try to make headway along this direction by introducing an executable language to specify

agents' *normative positions* and manage their changes as agents interact via speech acts [11].

In this paper we present a language that acts as a "machine language" for norms on top of which different, higher-level normative languages can 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 and states of affairs. In this way, normative positions can be further refined. For instance, picture a selling agent that is obliged to deliver a good satisfying some quality requirements before a deadline. Notice that both the quality requirements and the delivery deadline can be regarded as constraints that must be considered as part of the obligations. Thus, when the agent delivers the good satisfying all the constraints, we should regard the obligation as fulfilled. Notice too that since the deadline might eventually be changed, we also require the capability of modifying constraints at run-time. Hence, constraints are considered as first-class citizens in our language.

Although in this paper we restrict to a particular class of MASs, namely electronic institutions [12], our work sets the foundations to specify and implement open regulated MASs via norms.

The structure of this paper is as follows. In the next section we present desirable properties of normative languages. In section 3 we propose a simple normative language that covers all these requirements along with a sketch of an implementation of an interpreter. Section 4 summarises electronic institutions and explains how we capture normative positions of participating agents. We put our language to use by specifying the Dutch Auction protocol in section 5. In section 6 we contrast our approach with a sample of other contemporary work. Finally, we draw conclusions and outline future work in section 7.

2 Norm-Oriented MAS: Desiderata

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 in candidate languages.

Explicit Management of normative positions We take the stance that we cannot refer to agents' mentalistic notions, but only to their observable actions and their normative positions. Notice that as a result of agents' observable, social interactions, their normative positions [4] change. Hence, the first requirement of our language is to support the *explicit management* of agents' normative positions.

General purpose 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.

Pragmatic In a sense, we pursue a "machine language" for norms on top of which higher-level languages may be accommodated. 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 the language supports their encoding – this is 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.

Declarative 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 Rule Language for Norms

In this section we introduce a rule language for the explicit management of norms associated with a population of agents. Our rule-based language allow us to represent changes in an elegant way and also fulfils the requirement that a normative language should be declarative. The rules depict how normative positions change as agents interact with each other. We achieve greater flexibility, expressiveness and precision by allowing *constraints* [13] to be part of our rules – such constraints associate further restrictions with permissions, prohibitions and obligations.

The building blocks of our language are first-order terms (denoted as τ) and implicitly, universally quantified atomic formulae (denoted as α) without free variables . We shall make use of numbers and arithmetic functions to build terms; arithmetic functions may appear infix, following their usual conventions[3]. We also employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation with their usual meaning. 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 subset of atomic formulae:

Definition 1. *A constraint γ is an atomic formula of the form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.*

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

[3] We adopt Prolog's convention using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

Intuitively, a state of affairs is a set of atomic formulae. As we will show below, they can store the state of the environment[4], observable agent attributes and the normative positions of agents:

Definition 2. *A state of affairs $\Delta = \{\alpha_0, \ldots, \alpha_n\}$ is a a finite and possibly empty set of implicitly, universally quantified atomic formulae $\alpha_i, 0 \leq i \leq n, n \in \mathbb{N}$.*

Our rules 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. The grammar in Fig. 1 defines our rules, where x is a variable name and LHS^* is a LHS without set constructors (see below). The Us represent the updates: they add (via operator $\oplus$) or remove (via operator $\ominus$) atomic formulae αs. Furthermore, we make use of a special kind of term, called a *set constructor*, represented as $\{\alpha' \mid LHS^*\}$. This construct is useful when we need to refer to all α's for which LHS^* holds,e.g., $\{p(A,B) \mid A > 20 \wedge B < 100\}$ is the set of atomic formulae $p(A,B)$ such that $A > 20$ and $B < 100$.

$$\begin{aligned}
\mathsf{R} &::= LHS \rightsquigarrow RHS \\
LHS &::= LHS \wedge LHS \mid \neg LHS \mid \mathsf{Lit} \\
RHS &::= \mathsf{U} \bullet RHS \mid \mathsf{U} \\
\mathsf{Lit} &::= \alpha \mid x = \{\alpha' \mid LHS^*\} \\
\mathsf{U} &::= \oplus\alpha \mid \ominus\alpha
\end{aligned}$$

Fig. 1. Grammar for Rules

We need to refer to the set of constraints that belongs to a state of affairs. We call $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ the set of all constraints in Δ.

Definition 3. *Given a state of affairs Δ, relationship $constrs(\Delta, \Gamma)$ holds iff Γ is the smallest set such that for every $\gamma \in \Delta$ then $\gamma \in \Gamma$.*

In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation, as well as the concept of its application to a term [14]. We now define the semantics of our rules as relationships between states of affairs: rules map an existing state of affairs to a new state of affairs. We adopt the usual semantics of production rules, 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 via $\mathbf{s}^*$.

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

[4] We refer to the *state of the environment* as the set of atomic formulae that represent the aspects of the environment in a given point in time.

side match Δ and apply these substitutions to RHS (that is, $RHS \cdot \sigma_i$) in order to build Δ'.

Our rules are *exhaustively* applied on the state of affairs thus considering all matching atomic formulae. We thus need relationship $\mathbf{s}^*_l(\Delta, LHS, \Sigma)$ which obtains in $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$ *all* possible matches of the left-hand side of a rule:

Definition 5. *$\mathbf{s}^*_l(\Delta, LHS, \Sigma)$ holds, iff $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is the largest non-empty set such that $\mathbf{s}_l(\Delta, LHS, \sigma_i), 1 \leq i \leq n, n \in I\!N$, holds.*

We now define the semantics of the LHS of a rule:

Definition 6. *$\mathbf{s}_l(\Delta, LHS, \sigma)$ holds between state Δ, the left-hand side of a rule LHS and a substitution σ depending on the format of LHS:*

1. *$\mathbf{s}_l(\Delta, LHS \wedge LHS', \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma')$ and $\mathbf{s}_l(\Delta, LHS', \sigma'')$ hold and $\sigma = \sigma' \cup \sigma''$.*
2. *$\mathbf{s}_l(\Delta, \neg\, LHS, \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma)$ does not hold.*
3. *$\mathbf{s}_l(\Delta, \alpha', \sigma)$ holds iff $\alpha' \cdot \sigma \in \Delta$ and $constrs(\Delta, \Gamma)$ and $satisfiable(\Gamma \cdot \sigma)$ hold.*
4. *$\mathbf{s}_l(\Delta, \gamma, \sigma)$ holds iff $constrs(\Delta, \Gamma)$ and $satisfiable((\Gamma \cup \{\gamma\}) \cdot \sigma)$ hold.*
5. *$\mathbf{s}_l(\Delta, x = \{\alpha' \mid LHS'\}, \sigma)$ holds iff $\sigma = \{x/\{\alpha' \cdot \sigma_1, \ldots, \alpha' \cdot \sigma_n\}\}$ for the largest $n \in I\!N$ such that $\mathbf{s}_l(\Delta, \alpha' \wedge LHS', \sigma_i), 1 \leq i \leq n$*

Cases 1-3 depict the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction. Case 4 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 check satisfiability of constraints [5]. Case 5 specifies the semantics for *set constructors*: x is the set of atomic formulae that satisfy the conditions of the set constructor.

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

1. *$\mathbf{s}_r(\Delta, (\mathsf{U} \bullet RHS), \Delta')$ holds iff both $\mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1)$ and $\mathbf{s}_r(\Delta_1, RHS, \Delta')$ hold.*
2. *$\mathbf{s}_r(\Delta, \oplus\alpha', \Delta')$ holds iff $\Delta' = \Delta \cup \{\alpha'\}$.*
3. *$\mathbf{s}_r(\Delta, \oplus\gamma, \Delta') = \mathbf{true}$ iff $constrs(\Delta, \Gamma)$ and $satisfiable(\Gamma \cup \{\gamma\})$ hold and $\Delta' = \Delta \cup \{\gamma\}$.*
4. *$\mathbf{s}_r(\Delta, \ominus\alpha, \Delta')$ holds iff $\Delta' = \Delta \setminus \{\alpha\}$*

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 cater for the insertion of atomic formulae α' which do not conform to the syntax of constraints. Case 3 defines how a constraint is added to a state Δ: the new constraint is checked whether it can be satisfied with constraints Γ and then it is added to Δ'. Case 4 cater for the removal of atomic formulae.

We extend $\mathbf{s}^*$ to handle sequences of rules: $\mathbf{s}^*(\Delta_0, \langle \mathsf{R}_1, \ldots, \mathsf{R}_n \rangle, \Delta_n)$ holds iff $\mathbf{s}^*(\Delta_{i-1}, \mathsf{R}_i, \Delta_i), 1 \leq i \leq n$ hold.

[5] Our work builds on standard technologies for constraint solving – in particular, we have been experimenting with SICStus Prolog constraint satisfaction libraries.

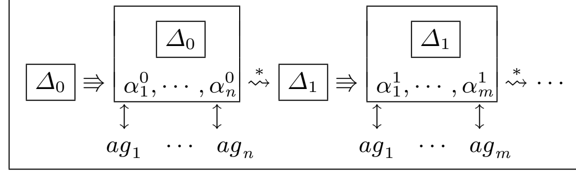

Fig. 2. Semantics as a Sequence of Δ's

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. Fig. 2 illustrates how this sequence can accommodate the intervention of agents sending/receiving messages. The diagram shows an initial state Δ_0 (possibly empty) that is offered (represented by "$\Rrightarrow$") to a set of agents $\{ag_1, \ldots, ag_n\}$. These agents exchange messages, adding a record (via "$\updownarrow$") $\{\alpha^0_1, \ldots, \alpha^0_n\}$ of these messages to Δ_0. After the agents add their utterances, then the rules are exhaustively applied (represented by "$\overset{*}{\rightsquigarrow}$") to $\Delta_0 \cup \{\alpha^0_1, \ldots, \alpha^0_n\}$. The resulting state Δ_1 is, on its turn, offered to agents, and so on.

3.1 Implementation

The semantics above provide a basis for the implementation of our rule interpreter. Although we have implemented it with SICStus Prolog we show such interpreter in Fig. 3 as a logic program, interspersed with built-in Prolog predicates; for easy referencing, we show each clause with a number on its left.

1. $\mathbf{s}^*(\Delta, Rules, \Delta') \leftarrow$
   $\texttt{findall}(\langle RHS, \Sigma\rangle, (\texttt{member}((LHS \rightsquigarrow RHS), Rules), \mathbf{s}^*_l(\Delta, LHS, \Sigma)), RHSs),$
   $\mathbf{s}'_r(\Delta, RHSs, \Delta')$
2. $\mathbf{s}^*_l(\Delta, LHS, \Sigma) \leftarrow \texttt{findall}(\sigma, \mathbf{s}_l(\Delta, LHS, \sigma), \Sigma)$
3. $\mathbf{s}_l(\Delta, (LHS \wedge LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma'), \mathbf{s}_l(\Delta, LHS', \sigma''), \texttt{union}(\sigma', \sigma'', \sigma)$
4. $\mathbf{s}_l(\Delta, \neg\, LHS, \sigma) \leftarrow \neg\, \mathbf{s}_l(\Delta, LHS, \sigma)$
5. $\mathbf{s}_l(\Delta, \alpha', \sigma) \leftarrow \texttt{member}(\alpha' \cdot \sigma, \Delta), constrs(\Delta, \Gamma), \texttt{satisfiable}(\Gamma \cdot \sigma)$
6. $\mathbf{s}_l(\Delta, \gamma, \sigma) \leftarrow constrs(\Delta, \Gamma), \texttt{satisfiable}([\gamma \mid \Gamma] \cdot \sigma)$
7. $\mathbf{s}_l(\Delta, x = \{\alpha' \mid LHS'\}, \{x/AllAlphas\}) \leftarrow \texttt{findall}(\alpha' \cdot \sigma, \mathbf{s}_l(\Delta, \alpha' \wedge LHS', \sigma), AllAlphas)$
8. $\mathbf{s}'_r(\Delta, [\,], \Delta') \leftarrow \Delta = \Delta'$
9. $\mathbf{s}'_r(\Delta, [\langle RHS, \Sigma\rangle \mid RHSs], \Delta') \leftarrow \mathbf{s}''_r(\Delta, RHS, \Sigma, \Delta''), \mathbf{s}'_r(\Delta'', RHSs, \Delta')$
10. $\mathbf{s}''_r(\Delta, RHS, [\,], \Delta') \leftarrow \Delta = \Delta'$
11. $\mathbf{s}''_r(\Delta, RHS, [\sigma \mid \Sigma], \Delta') \leftarrow \mathbf{s}_r(\Delta, RHS \cdot \sigma, \Delta''), \mathbf{s}''_r(\Delta'', RHS, \Sigma, \Delta')$
12. $\mathbf{s}_r(\Delta, (\mathsf{U} \bullet RHS), \Delta') \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1), \mathbf{s}_r(\Delta_1, RHS, \Delta')$
13. $\mathbf{s}_r(\Delta, \oplus\alpha', [\alpha' \mid \Delta]) \leftarrow$
14. $\mathbf{s}_r(\Delta, \ominus\alpha, \Delta') \leftarrow \texttt{delete}(\Delta, \alpha, \Delta')$
15. $\mathbf{s}_r(\Delta, \oplus\gamma, [\gamma \mid \Delta]) \leftarrow constrs(\Delta, \Gamma), \texttt{satisfiable}([\gamma \mid \Gamma])$

Fig. 3. An Interpreter for Rules

Clause 1 contains the top-most definition: given an existing Δ and a set of rules $Rules$, it obtains the next state Δ' by finding all those rules in $Rules$ (picked by the `member` built-in) whose LHS holds in Δ (checked via the auxiliary

definition $\mathbf{s}_l^*$). This clause then uses the RHS of those rules with their respective sets of substitutions Σ as the arguments of $\mathbf{s}_r'$ to finally obtain Δ'. Clause 2 implements $\mathbf{s}_l^*$: it finds all the different ways that the left-hand side LHS of a rule can be matched in Δ – the individual σ's are stored in sets Σ of substitutions, as a result of the `findall/3` execution. Clauses 8 and 9 show how $\mathbf{s}_r'$ computes the new state from a list *RHSs* of pairs $\langle RHS, \Sigma \rangle$ (obtained in the second body goal of clause 1): it picks out each pair $\langle RHS, \Sigma \rangle$ and uses $\mathbf{s}_r''$ (clauses 10 and 11) to compute each intermediate state of affairs after applying the RHS to Δ via predicate $\mathbf{s}_r$ for all the substitutions in Σ. Clauses 3-7 and 12-15 are, respectively, adaptations of the cases depicted in Def. 6 and Def. 7.

4 Electronic Institutions

Our work extends *electronic institutions* (EIs) [12], 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:

Definition 8. *Illocutions* I *are terms* $p(ag, r, ag', r', \tau, t)$ *where* p *is an illocutionary particle (*e.g.*,* ask*); ag, ag' are agent identifiers; r, r' are role labels; τ is a term with the actual content of the message and $t \in I\!N$ is a time stamp.*

We shall refer to illocutions that may have uninstantiated (free) variables as *illocution schemes*, denoted by $\bar{\mathsf{I}}$.

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 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 $\bar{\mathsf{I}}$ 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 the state of the environment, all observable attributes of agents and all obligations, permissions and prohibitions associated with the agents that constitute their normative positions.

We differentiate seven kinds of atomic formulae in our institutional states Δ, with the following intuitive meanings:

1. $oav(o, a, v)$ – object (or agent) o has an attribute a with value v.
2. $att(s, w, \mathsf{I})$ – an agent attempted to get illocution I accepted at state w of scene s.
3. $utt(s, w, \mathsf{I})$ – I was accepted as a legal utterance at w of s.
4. $ctr(s, w, t_s)$ – the execution of scene s reached state w at time t_s.
5. $obl(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ ought to be uttered at w of s.
6. $per(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *permitted* to be uttered at w of s.
7. $prh(s, w, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *prohibited* at w of s.

We only allow fully ground attributes, illocutions and state control formulae (cases 1-4 above) to be present[6]; however, in formulae 5-7 s and w may be variables and $\bar{\mathsf{I}}$ may contain variables. We shall use formulae 4 to represent state change in a scene in relation to a global clock. We shall use formulae 5–7 above to represent normative positions of agents within EIs.

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 the generality claimed on section 2 on our approach as different deontic relationships can be forged, as we show below. We can confer different grades of enforcement on EIs . On the one hand, we can transform only legal attempts into accepted utterances:

$$\begin{pmatrix} att(S, W, I) \wedge \\ per(S, W, I) \wedge \neg\, prh(S, W, I) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ominus att(S, W, I) \bullet \\ \oplus utt(S, W, I) \end{pmatrix} \tag{1}$$

This rule states that if an agent attempts to say something and it is permitted and not prohibited, then that attempt becomes a (confirmed) utterance. On the other hand, we can allow agents to do certain illegal actions under more harsh penalties:

$$\begin{array}{l} \begin{pmatrix} att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \wedge \\ Ag_1 \neq Ag_2 \wedge Ag_1 \neq Ag_3 \wedge Ag_2 \neq Ag_3 \end{pmatrix} \\ \rightsquigarrow \\ \begin{pmatrix} \ominus att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \bullet \\ \oplus utt(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \end{pmatrix} \end{array} \tag{2}$$

The rule above states that if an agent attempts to reveal to Ag_2 (secret) information about agent Ag_3, it is accepted without taking into account if it is forbidden or not. In both cases (rules 1 and 2), we can punish agents that violate prohibitions. Although we can address all forbidden utterances if we use a variable as the third parameter of *att* and *prh*, the following rule punishes only the revelation of beliefs of third parties:

$$\begin{array}{l} \begin{pmatrix} att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \wedge \\ Ag_1 \neq Ag_2 \wedge Ag_1 \neq Ag_3 \wedge Ag_2 \neq Ag_3 \wedge \\ prh(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \wedge \\ oav(Ag_1, rep, V_{Rep}) \wedge (V'_{Rep} = V_{Rep} - 10) \end{pmatrix} \\ \rightsquigarrow \\ \left(\ominus oav(Ag_1, rep, V_{Rep}) \bullet \oplus oav(Ag_1, rep, V'_{Rep}) \right) \end{array}$$

The rule above states that when agent Ag_1 tries to reveal to Ag_2 information about agent Ag_3, it gets punished. Notice that agents can be punished by decreasing the value of any of their observable attributes. But only for exemplifying purposes, we use here an attribute called *rep* (for reputation) that models in which degree an agent is norm compliant. In the example, the punish consists in decreasing the trust of agents to share information with Ag_1, that is, the value of Ag_1's reputation is decreased by 10.

[6] We allow agents to utter whatever they want (via *att* formulae). However, the illegal utterances may be discarded and/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.

5 Example: The Dutch Auction

We now illustrate the pragmatics of our norm-oriented language, as required in section 2, by specifying, with the rules of Fig. 4, the auction protocol for a fish market as described in [15]. In the fish market several scenes [12] take place simultaneously, at different locations, but with some causal continuity. The principal scene is the auction itself, where 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: 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; 2. With a chosen good, the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, previously fixed by a sellers' admitter, as long as these price quotations are above a *reserve price* previously defined by the seller; 3. For each price the auctioneer calls, several situations might arise during the open round described below. 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.

5.1 Proposed Solution

I. **Multiple bids** – it 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% of the collision price). Notice that X will hold all the utterances at scene *dutch* and state w_4 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 no further offers whose time-stamps 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;

II. **One bid/winner determination** – If only one bid has occurred during the current bidding round and the credit of the bidding agent is greater than or equal to the price of the good in auction, the rule adds the obligation for the auctioneer to inform all the buyers about the sale.

III. **Prevention** – It prevents agents from issuing bids they cannot afford (their credit is insufficient) and states that if agent Ag's credit is less than P (the last offer the auctioneer called for item It, at state w_3 of scene *dutch*), then agent Ag is prohibited to bid.

$$(X = \{ \alpha_0 | \alpha_1 \wedge \neg (\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge | X | > 1) \rightsquigarrow (\oplus\alpha_3 \bullet \oplus\alpha_4 \bullet \oplus(P_2 > P * 1.2))$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, collision(It, P), T_2)) \\ \alpha_4 = obl(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \tag{I}$$

$$\begin{pmatrix} X = \{ \alpha_0 | \alpha_1 \wedge \neg (\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge \\ | X | = 1 \wedge oav(A_1, credit, C) \wedge C \geq P \end{pmatrix} \rightsquigarrow (\oplus\alpha_3)$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, sold(It, P, A_1), T_4)) \end{cases} \tag{II}$$

$$(\alpha_0 \wedge \neg (\alpha_1 \wedge T2 > T) \wedge oav(Ag, credit, C) \wedge C < P) \rightsquigarrow (\oplus\alpha_2)$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T_2)) \\ \alpha_2 = prh(dutch, w_4, inform(A, buyer, Au, auct, bid(It, P_2), T_3)) \end{cases} \tag{III}$$

$$\begin{pmatrix} X = \{ \alpha_0 | \alpha_1 \wedge \neg (\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1 \} \wedge \\ | X | = 1 \wedge oav(A_1, credit, C) \wedge C < P \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ominus oav(A_1, credit, C) \bullet \\ \oplus oav(A_1, credit, C_2) \bullet \oplus\alpha_3 \bullet \\ \oplus(C_2 = C - P * 0.1) \bullet \oplus(P_2 = P * 1.2) \end{pmatrix}$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \tag{IV}$$

$$\begin{pmatrix} ctr(dutch, w_5, T_n) \wedge \alpha_0 \wedge \neg (\alpha_1 \wedge T_2 > T) \wedge \\ timeout(dutch, w_4, w_5, T_3) \wedge T_3 > T \wedge \\ oav(IT, reservation_price, RP) \wedge \\ oav(IT, decrement_rate, DR) \wedge RP < P - DR \end{pmatrix} \rightsquigarrow \begin{pmatrix} \oplus\alpha_2 \bullet \\ \oplus(P2 = P - DR) \end{pmatrix}$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T)) \\ \alpha_1 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T_2)) \\ \alpha_2 = obl(dutch, w5, inform(Au, auct, all, buyer, offer(IT, P2), T_4)) \end{cases} \tag{V}$$

$$\begin{pmatrix} ctr(dutch, w_5, T_n) \wedge \alpha_0 \wedge \neg (\alpha_1 \wedge T_2 > T) \wedge \\ timeout(dutch, w_4, w_5, T_3) \wedge T_3 > T \wedge oav(It, reservation_price, RP) \wedge \\ oav(It, decrement_rate, DR) \wedge RP \geq P - DR \end{pmatrix} \rightsquigarrow (\oplus\alpha_2)$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_2 = obl(dutch, w_5, inform(Au, auct, all, buyer, withdrawn(It), T_3)) \end{cases} \tag{VI}$$

Fig. 4. Rules for the Dutch Auction Protocol

IV. **Punishment** – It punishes agents when issuing a winning bid they cannot pay for. More precisely, the rule punishes an agent A_1 by decreasing his credit of 10% of the value of the good being auctioned. The *oav* predicate on the LHS of the rule represents the current credit of the offending agent. The rule also adds an obligation for the auctioneer to restart the bidding round and the constraint that the new offer should be greater than 120% of the old price.

V. **No bids/New Price** – It 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 5 checks that the current scene state is w_5, 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.

VI. **No bids/withdrawal** – It 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 6 checks that the current institutional state is w_5, 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 the price of item It.

6 Comparison with Other Normative Languages

In this section we compare our proposal with other normative languages in the literature. We concentrate on different approaches, explaining how we can capture a wide range of normative notions from these formalisms using our rule language. In doing so, we can provide an *implementation* for some of these formalisms.

A norm from [16] is composed of several parts: the norm condition is the declarative description of the norm and the context in which it applies; the *violation condition* (a formula defining when the norm is violated); the *detection mechanism* describing the mechanisms that can be used for detecting violations; 3) the *sanctions* defined as actions to punish the agents' violation of the norm; and the *repairs* (a set of actions that are used for recovering the system after the occurrence of a violation). Through the condition (IF) and temporal operators (BEFORE and AFTER), which are considered optional, norms can be made applicable only to certain situations. Temporal operators can be applied to a deadline or to an action or predicate.

Norms as defined in [16] can be translated into our rules by specifying the violation conditions on the LHS and sanctions and repairs on the RHS. Since we consider illocutions as the only actions that can be performed in an electronic institution, actions need to be translated into illocutions uttering that the action has been done. We call this operation *contextualisation*. In general, the translation of the norms of [16] into our rules is straightforward. The permission of an action is translated as a rule that converts the attempt to utter illocution, *i.e.*, $att(S, W, I)$, into the illocution being uttered, *i.e.*, $utt(S, W, I)$. The prohibition of an action can be translated into a rule that ignores the attempt to utter the illocution, and, optionally, a sanction to the violation can be imposed. The obligation of an action needs to be translated into two rules, *viz.*, a rule to sanction an agent when it does not fulfil an obligation (*i.e.*, not uttering the expected illocution at the right scene and state), and a rule to remove the obligation once it is fulfilled. The translation of temporal clauses (BEFORE and AFTER) can be achieved by adding to the LHS of the rule the condition that the time in which the attempt is done has to be less (or greater) than the deadline.

Although the work in [3] proposes a framework that covers several topics of normative multi-agent systems we shall focus on its definition of norm, in which *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. Notice that a norm must always have addressees, normative goals and a context; *rewards* and *punishments* are disjoint sets, and *context* and *exceptions* too.

A norm from [3] can be translated into the following rule schema to detect its violation:

$$(\mathit{context} \ \wedge \neg \ \mathit{exception} \wedge \neg \ \mathit{goal}') \rightsquigarrow \mathit{punishments}$$

where *context* and *exception* are predicates obtained through contextualisation[7] for specifying the context and exceptions mentioned in the norm, goal' is the contextualised normative goal (which includes the addressee and possible beneficiaries). Component *punishments* are contextualised actions obtained from the norm. This rule captures that in a particular context which is not an exception of the norm and whose goal has not yet been fulfilled the actions defined by *punishments* should be executed.

Rewards can also be specified via the rule schema:

$$(\mathit{context} \ \wedge \neg \ \mathit{exception} \wedge \mathit{goal}') \rightsquigarrow \mathit{rewards}$$

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.

Event calculus is used in [6] for the specification of protocols. Event calculus is a formalism to represent reasoning about actions or events and their effects in a logic programming framework and is based on a many-sorted first-order predicate calculus. Predicates that change with time are called *fluents*. In [6] obligations, permissions, empowerments, capabilities and sanctions are formalised by means of fluents – prohibitions are not formalised in [6] as a fluent since they assume that every action not permitted is forbidden by default. If we translate all the *holdsAt* predicates into *utt* predicates, we can translate the obligations and permissions of [6] by including the rest of conditions in the LHS of the normative rules. However, since there is no concrete definition of norm, we cannot state that the approach in [6] is fully translatable into our rules.

Although event calculus models time, the deontic fluents specified in the example of [6] are not enough to inform an agent about all types of duties. For

[7] Recall that contextualisation is the process of transforming actions into illocutions stating that actions have been brought about.

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.

In [7] we find a proposal to represent norms via 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. That proposal uses the technique of formula progression from the TLPlan planning system to monitor social expectations until they are fulfilled or violated.

Intuitively, our rules capture formulae $\mathsf{AG}^{+}(LHS \rightarrow \mathsf{X}^{+}RHS)$ where LHS and RHS are atomic formulae without temporal operators. As we build the next state of affairs by applying the operations on the RHS of the fired rules, we cannot use any other temporal operator in the RHS of our rules. Furthermore, since our state of affairs has non-monotonic features we cannot reason over the past of any formulae. We can only do it with 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 on the LHS of the hyMITL rule: $\mathsf{X}^{-}\phi$ is intuitively equivalent to $ctr(S, W, T_s) \wedge \phi(T_0) \wedge T_0 = T_s - 1$. Moreover, we can also translate the U^{+} operator when it is used in the RHS of the hyMITL rule: $\phi\ \mathsf{U}^{+}\psi$ is roughly equivalent to $\psi \rightsquigarrow \ominus\phi$. Although we cannot use all the temporal operators on the RHS of our rules, we can obtain equivalent results by imposing certain restrictions in the set of rules. $\mathsf{F}^{+}\phi$ can be achieved if $\oplus\phi$ appears on 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.

In [17] the language Social Integrity Constraints (SIC) is proposed. This language's constructs check whether some events have occurred and some conditions hold to add new expectations, optionally with constraints. Although syntactically their language is very similar to ours, they are semantically different. Different from their use of abduction and Constraint Handling Rules (CHR) to execute their expectations, we use a forward chaining approach. Despite the fact that expectations they use are quite similar to obligations and they mention how expectations are treated, that is, what happens when an expectation is fulfilled or when it is not, and state the possibility of SICs being violated, no mechanism to regulate agents' behaviour like the punishment of offending agents or repairing actions are offered.

The work in [8] proposes the Object Constraint Language (OCL) for the specification of artificial institutions. The example of this work commits an auctioneer not to declare a price lower than the agreed reservation price. As shown in section 5, we can also express (rule VI) the case that the auctioneer is obliged to withdraw the good when the call price becomes lower than the reservation price. As for [8], we cannot perform an exhaustive analysis of the language because neither the syntax nor the semantics are specified.

The approach in [18] uses Answer Set Programming (ASP) [19] for the specification and analysis of agent-based social institutions. They state that ASP

overcomes many Prolog limitations since, instead of calculating only the first possible solution, it provides all answers to a query. Although ASP is suitable for institution analysis, it may not be so efficient as required for institution execution since only one answer is needed, *viz.*, the next state of affairs.

As for institution modelling, they include institutional facts and actions, permissions, prohibitions, obligations (only) with deadline, violations and institutional power. The latter, not included in our EI model: it specifies that a certain agent is empowered to perform a specified institutional action in a given institution. However, they do not include the possibility of rewarding for norm compliance nor managing other constraints than deadlines.

The work in [9] reports on the translation of the normative language presented in [16] into Jess rules 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 bears strong similarities with the work reported in [20] where norms are represented as rules of a production system. We notice that our rules can express their notions of contracts and their monitoring (*i.e.*, fulfilment and violation of obligations). However, in [20] constraints can only be used to depict the left-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.

After analysing all these approaches we have found some norm patterns that they have in common. Norms can be conditional or can have temporal constraints, that is, they establish relationships between time-points or events or they hold periodically. Our rules can capture the patterns from rather disparate formalisms, thus fulfilling the requirement of general purpose mentioned in section 2.

7 Conclusions and Future Work

In this paper we have introduced a formalism for the explicit management of the normative position of agents in electronic institutions. Ours is a rule language in which constraints can be specified and changed at run-time, conferring expressiveness and precision on our constructs. The semantics of our formalism defines a kind of production system in which rules are exhaustively applied to a

state of affairs, leading to the next state of affairs. The normative positions are updated via rules, depending on the messages agents send.

Our formalism addresses the points of a desiderata for normative languages introduced in section 2. We have explored our proposal in this paper by specifying a version of the Dutch Auction protocol. We illustrate how our language can provide other (higher-level) normative languages with a computational model (*i.e.*, an *implementation*) thus making it possible for normative languages proposed with more theoretical concerns in mind to become executable.

However, we notice that although our implementation directly captures the proposed formal semantics, it is not as efficient as other implementations for rule-based systems, such as the Rete algorithm [21].

As for future work, we would like to overcome the efficiency issue by providing an implementation based on the Rete algorithm.

We would also like to generalise our language to cope with arbitrary actions, rather than just speech acts among agents – this would allow our work to address any type of open multi-agent system. We would also like to improve the semantics of the language in order to support the use of temporal operators for the management of time.

Our semantics describe a transition system similar to the one presented in [22] – we would like to carry out a careful comparison between that work and our operational semantics.

An interesting avenue of investigation is to endow agents with reasoning abilities over our rules. Such reasoning, possibly using resource-bounded forward and backward chaining mechanisms, would allow agents to anticipate the effects of their actions, that is, the punishments or rewards for, respectively, norm violation and norm compliance

We also want to investigate the verification of norms (along the lines of our work in [23]) expressed in our rule language, with a view to detecting, for instance, obligations that cannot be fulfilled, prohibitions that will prevent progress, inconsistencies and so on. We are currently investigating tools to help engineers preparing their rules – these are norm editors that will support the design and verification of norm-oriented electronic institutions. Our language is very suitable to be added with a typing system and associated type checking mechanism thus providing additional support for designers when preparing their rules.

Acknowledgements – This work was partially funded by the Spanish Science and Technology Ministry as part of the Web-i-2 project (TIC-2003-08763-C02-00). García-Camino enjoys an I3P grant from the Spanish Council for Scientific Research (CSIC).

References

1. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons, Chichester, UK (2002)

2. Dignum, F.: Autonomous Agents with Norms. Artificial Intelligence and Law **7**(1) (1999) 69–79
3. López y López, F.: Social Power and Norms: Impact on agent behaviour. PhD thesis, Univ. of Southampton (2003)
4. Sergot, M.: A Computational Theory of Normative Positions. ACM Trans. Comput. Logic **2**(4) (2001) 581–622
5. Shoham, Y., Tennenholtz, M.: On Social Laws for Artificial Agent Societies: Off-line Design. Artificial Intelligence **73**(1-2) (1995) 231–252
6. Artikis, A., Kamara, L., Pitt, J., Sergot, M.: A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. Volume 3476 of LNCS. Springer-Verlag (2005)
7. Cranefield, S.: A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. Technical Report 2005/01, Univ. of Otago (2005)
8. Fornara, N., Viganò, F., Colombetti, M.: An Event Driven Approach to Norms in Artificial Institutions. In: AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM), Utrecht (2005)
9. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing Norms in Electronic Institutions. In: Procs. 4th AAMAS. (2005)
10. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: A Distributed Architecture for Norm-Aware Agent Societies. In: Procs. Int'l Workshop on Declarative Agent Languages & Technologies (DALT 2005), New York, USA. Volume 3904 of LNAI. Springer-Verlag, Berlin (2006)
11. Searle, J.: Speech Acts, An Essay in the Philosophy of Language. Cambridge University Press (1969)
12. Esteva, M.: Electronic Institutions: from Specification to Development. PhD thesis, Universitat Politècnica de Catalunya (UPC) (2003) IIIA monography Vol. 19.
13. Jaffar, J., Maher, M.J.: Constraint Logic Programming: A Survey. Journal of Logic Progr. **19/20** (1994) 503–581
14. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, U.S.A. (1990)
15. Noriega, P.: Agent-Mediated Auctions: The Fishmarket Metaphor. PhD thesis, Universitat Autònoma de Barcelona (UAB) (1997) IIIA monography Vol. 8.
16. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing Norms in Multiagent Systems. Volume 3187 of LNAI. Springer-Verlag (2004)
17. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and Verification of Agent Interactions using Integrity Social Constraints. Technical Report DEIS-LIA-006-03, University of Bologna (2003)
18. Cliffe, O., De Vos, M., Padget, J.: Specifying and Analysing Agent-based Social Institutions using Answer Set Programming. In: AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM), Utrecht (2005)
19. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge Press (2003)
20. Lopes Cardoso, H., Oliveira, E.: Towards an Institutional Environment using Norms for Contract Performance. Volume In press of LNAI., Springer-Verlag (2005)
21. Forgy, C.: Rete: a fast algorithm for the many pattern/many object pattern match problem". Artificial Intelligence **19**(1) (1982) 17–37
22. Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. Journal of Logic Programming **17** (1993) 301–321
23. Vasconcelos, W.W.: Norm Verification and Analysis of Electronic Institutions. Volume 3476 of LNAI. Springer-Verlag (2004)