Norm-Oriented Programming of Electronic Institutions*

A. García-Camino,
J.-A. Rodríguez-Aguilar, C. Sierra
IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
{andres, jar, sierra}@iiia.csic.es

W. Vasconcelos
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.

Categories and Subject Descriptors

I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Law*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multi-agent systems*

General Terms

Languages

Keywords

Norms, Electronic institutions, Multi-agent system programming

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 self-interested agents defined with different languages by several designers. 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 [13, 2, 3, 8]. There is a wealth of socio-philosophical and logic-theoretical literature on the subject of norms (*e.g.*, [11, 12]), 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. [1, 6, 7]).

A normative position [11] is the "social burden" associated with individual agents, namely, 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 [10].

In this paper we present a language that acts as a "machine language" for norms on top of which 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. 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 [4], our work sets the foundations to specify and implement open regulated MASs via norms.

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. As a result of agents' observable, social interactions, their normative positions [11] change. Hence, the first requirement of our language is to support the *explicit management* of agents' normative positions.

General purpose. 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. We pursue a "machine language" for norms on top of which higher-level languages can 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

*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).

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

their encoding[1]. In this way, we shall guarantee the expressiveness of our language, and also address 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.

2. A RULE LANGUAGE FOR NORMS

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[2]. 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: 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.

Intuitively, a state of affairs is a set of atomic formulae. As we will show below, they can store the state of the environment[3], observable agent attributes and the normative positions of agents: a state of affairs $\Delta = \{\alpha_0, \ldots, \alpha_n\}$ is a a finite and possibly empty set of implicitly, universally quantified atomic formulae.

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 $B < 100$.

R	::=	*LHS* $\rightsquigarrow$ *RHS*
LHS	::=	*LHS* $\wedge$ *LHS* $\mid \neg$(*LHS* $\wedge$ *LHS*) $\mid$ Lit
RHS	::=	U $\wedge$ *RHS* $\mid$ U
Lit	::=	$\alpha \mid \neg\alpha \mid x = \{\alpha' \mid LHS^*\}$
U	::=	$\oplus\alpha \mid \ominus\alpha$

Figure 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. 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 [5].

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 $\mathbf{s}^*$: $\mathbf{s}^*(\Delta, LHS \rightsquigarrow RHS, \Delta')$ holds iff $\mathbf{s}_l^*(\Delta, LHS, \{\sigma_1, \ldots, \sigma_n\})$ and $\mathbf{s}_r(\Delta, RHS \cdot \sigma_i, \Delta'), 1 \leq i \leq n, n \in I\!N$, hold. That is, two states of affairs Δ and Δ' are related by a rule *LHS* $\rightsquigarrow$ *RHS* iff we obtain all different substitutions $\{\sigma_1, \ldots, \sigma_n\}$ that make the left-hand 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: $\mathbf{s}_l^*(\Delta, LHS, \Sigma)$ 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: $\mathbf{s}_l(\Delta, LHS, \sigma)$ 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.[4] Case 5 specifies the semantics for *set constructors*: x is the set of atomic formulae that satisfy the conditions of the set constructor.

We now define the semantics of the *RHS* of a rule: 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. $\mathbf{s}_r(\Delta, (\mathsf{U} \wedge RHS), \Delta')$ 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 caters 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 caters for the removal of atomic formulae.

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

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

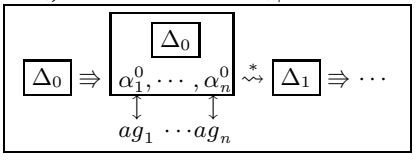

Figure 2: Semantics as a Sequence of Δ's

[1] Work available in `http://www.iiia.csic.es/~andres/NOPLforEIs.pdf`

[2] 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.

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

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

(via "$\updownarrow$") $\{\alpha_1^0, \ldots, \alpha_n^0\}$ of these messages to Δ_0. After the agents add their utterances, then the rules are exhaustively applied (represented by "$\overset{*}{\rightsquigarrow}$") to $\Delta_0 \cup \{\alpha_1^0, \ldots, \alpha_n^0\}$. The resulting state Δ_1 is, on its turn, offered to agents, and so on.

Our work extends *electronic institutions* (EIs) [4], 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. Illocutions I are terms $p(ag, r, ag', r', \tau, t)$ where p is an illocutionary particle (*e.g., inform,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 that we employ here is that of a *scene*. Scenes offer means to break down larger protocols into smaller ones with specific purposes.

We differentiate seven kinds of atomic formulae in our state of affairs 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[5]; 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 relationship with global time passing. We shall use formulae 5–7 above to represent normative positions of agents within EIs.

Thus, we can write rules for prohibiting (or permitting) actions by default, for permitting (or prohibiting) certain actions and for choosing which actions will be prevented or only sanctioned. For example, in the context of a Dutch auction [9], agents can be prevented from making unsupported bids or only sanctioned:

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

where

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

The rule above prevents agents from issuing bids in a Dutch auction they cannot afford (their credit is insufficient). It 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.

$$\left(X = \{\ \alpha_0 \mid \alpha_1 \wedge \neg\,(\alpha_2 \wedge T_2 > T_1) \wedge T_0 > T_1\ \} \wedge \begin{array}{c} \\ |\,X\,| = 1 \wedge oav(A_1, credit, C) \wedge C < P \end{array} \right) \rightsquigarrow$$

$$\left(\begin{array}{c} \ominus oav(A_1, credit, C) \wedge \\ \oplus oav(A_1, credit, C_2) \wedge \oplus\alpha_3 \wedge \\ \oplus(C_2 = C - P * 0.1) \wedge \oplus(P_2 = P * 1.2) \end{array} \right)$$

where

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

The previous rule punishes agents when issuing a winning bid they cannot pay for. More precisely, the rule punishes agent A_1 by decreasing his credit a 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 along with the constraint that the new offer should be greater than 120% the old price.

These examples illustrate how our language addresses the explicit management of normative positions and the pragmatic concerns raised in the desiderata.

[5] 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 semantics of the specified deontic notions. The utt formulae are thus *confirmations* of the att formulae that turn illocutions into legal.

3. 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 above.

We would 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 along the lines of [6].

4. REFERENCES

[1] 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, 2005.

[2] R. Axelrod. *The complexity of cooperation: agent-based models of competition and collaboration*. Princeton studies in complexity. Princeton University, New Jersey, 1997.

[3] F. Dignum. Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[4] M. Esteva. *Electronic Institutions: from Specification to Development*. PhD thesis, Universitat Politècnica de Catalunya (UPC), 2003. IIIA monography Vol. 19.

[5] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, U.S.A., 1990.

[6] A. García-Camino, P. Noriega, and J. A. Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *Procs. 4th AAMAS*, 2005.

[7] 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.

[8] F. López y López. *Social Power and Norms: Impact on agent behaviour*. PhD thesis, Univ. of Southampton, June 2003.

[9] P. Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona (UAB), 1997. IIIA monography Vol. 8.

[10] J. Searle. *Speech Acts, An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[11] M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.

[12] Y. Shoham and M. Tennenholtz. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73(1-2):231–252, 1995.

[13] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, Feb. 2002.