A Distributed Architecture for Norm-Aware Agent Societies

A. García-Camino[1], J. A. Rodríguez-Aguilar[1], C. Sierra[1], and W. Vasconcelos[2]

[1]Institut d'Investigació en Intel·ligència Artificial, CSIC
Campus UAB 08193 Bellaterra, Catalunya, Spain
{andres, jar, sierra}@iiia.csic.es
[2]Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
wvasconcelos@acm.org

Abstract. We propose a distributed architecture to endow multi-agent systems with a social layer in which normative positions are explicitly represented and managed via rules. Our rules operate on a representation of the states of affairs of a multi-agent system. We define the syntax and semantics of our rules and an interpreter; we achieve greater precision and expressiveness by allowing *constraints* to be part of our rules. We show how the rules and states come together in a distributed architecture in which a team of administrative agents employ a tuple space to guide the execution of a multi-agent system.

1 Introduction

Norms (*i.e.*, obligations, permissions and prohibitions) capture an important aspect of heterogeneous multi-agent systems (MASs) – they constrain and influence the behaviour of individual agents [1–3] as they interact in pursuit of their goals. In this paper we propose a distributed architecture built around an explicit model of the norms associated with a society of agents, consisting of:

- an information model storing the normative positions of MASs' individuals.
- a rule-based representation of how normative positions are updated during the execution of a MAS.
- a distributed architecture with a team of administrative agents to ensure normative positions are complied with and updated.

A normative position [4] is the "social burden" associated with an agent, that is, their obligations, permissions and prohibitions. We show in Fig. 1 our proposal and how its components fit together. Our architecture provides a *social layer* for multi-agent systems specified via electronic institutions (EI, for short) [5]. EIs specify the kinds and order of interactions among software agents with a view to achieving global and individual goals – although our study here concentrates on EIs we believe our ideas can be adapted to alternative frameworks. In our diagram we show a tuple space in which information models $\Delta_0, \Delta_1, \ldots$ are stored – these models are called *institutional states* (ex-

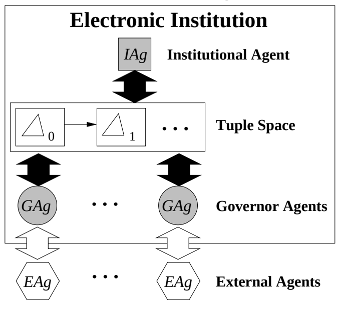

Fig. 1: Proposed Architecture

plained in Section 3) and contain all norms and other information that hold in specific points of time during the EI enactment.

The normative positions of agents are updated via *institutional rules* (described in Section 4). These are constructs of the form $\mathsf{LHS} \rightsquigarrow \mathsf{RHS}$ where LHS describes a condition of the current information model and RHS depicts how it should be updated, giving rise to the next information model. Our architecture is built around a shared tuple space [6] – a kind of blackboard system that can be accessed asynchronously by different administrative agents. In our diagram our administrative agents are shown in grey: the institutional agent updates the institutional state using the institutional rules; the governor agents work as "escorts" or "chaperons" to the external, heterogeneous software agents, writing onto the tuple space the messages to be exchanged.

In the next Section we introduce electronic institutions. In Section 3 we introduce our information model: the institutional states. In Section 4 we present the syntax and semantics of our institutional rules and how these can be implemented as a logic program; in that section we also give practical examples of rules. We provide more details of our architecture in Section 5. In Section 6 we contrast our proposal with other work and in Section 7 we draw some conclusions, discuss our proposal, and comment on future work.

1.1 Preliminary Concepts

We need to initially define some basic concepts. Our building blocks are first-order terms (denoted as T) and implicitly universally quantified atomic formulae (denoted as A) without free variables. We make use of numbers and arithmetic functions to build terms; arithmetic functions may appear infixed, following their usual conventions. We adopt Prolog's convention [7] and use strings starting with capital letters to represent variables and strings starting with lowercase letters 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 with their usual meaning. Atomic formulae with arithmetic relations represent *constraints* on their variables:

Def. 1. *A constraint* C *is of the form* $\mathsf{T} \lhd \mathsf{T}'$, *where* $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

We shall denote as B those atomic formulae that are *not* constraints.

2 Electronic Institutions

Our work expands *electronic institutions* (EIs) [5], 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. We define below the class of illocutions we aim at:

Def. 2. *Illocutions* 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 and* $T \in I\!N$ *is a time stamp.*

We shall refer to illocutions that may have uninstantiated variables as *illocution schemes*, denoted by $\bar{\mathsf{I}}$. 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 to abstract away from individuals.

Another important concept in EIs we employ here is that of a *scene*. Scenes are 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 EI is specified as a set of scenes connected by transitions – these are points where agents synchronise their movements between scenes [5].

An EI specification can be used to *synthesise* agents that conform to the specification [8]. The synthesised agents are called *governor agents*: although correct (that is, we guarantee they will follow the protocol since we provide their definition), they cannot make decisions on which messages to send (if there are different choices) nor on which values any unspecified variable ought to have. These choices should be made by *external agents* – these are heterogeneous agents that connect to a dedicated governor agent. The governor agent ensures the protocol is followed, whereas the external agent makes decisions as to which message to send (if there is a choice of messages) and any particular values illocutions ought to have.

Our architecture adds a social layer to the governor agents/external agents pairing. Although all illocutions of a protocol are *permitted* some of them may be deemed *inappropriate* in certain circumstances. For instance, although a protocol contemplates agents leaving a virtual auction room at any point, it may be inappropriate to do so if the agent has not yet paid what it owes. Our norms further restrict the expected behaviour of agents, prohibiting them from uttering an illocution or adding constraints on the values of variables of illocutions. Norms can be triggered by events involving any number of agents and their effects must persist until they are fulfilled or retracted by a rule.

3 Institutional States

Our states of affairs are called *institutional states*. Intuitively, they store global information on the enactment of an EI, such as utterances, agents' attributes, and so on, also keeping a record of all obligations, permissions and prohibitions associated with the agents. They are represented as a set of atomic formulae:

Def. 3. *An institutional state* $\Delta = \{\mathsf{A}_0, \ldots, \mathsf{A}_n\}$ *is a a finite and possibly empty set of implicitly universally quantified atomic formulae* $\mathsf{A}_i, 0 \leq i \leq n$*, without free variables.*

We differentiate five kinds of atomic formulae in our institutional states:

1. ground formulae $att(s, w, \mathsf{I})$ – I was attempted at state w of scene s.
2. ground formulae $utt(s, w, \mathsf{I})$ – I was a legal utterance at state w of scene s.
3. $obl(S, W, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ ought to be uttered at state W of scene S.
4. $per(S, W, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *permitted* to be uttered at state W of scene S.
5. $prh(S, W, \bar{\mathsf{I}})$ – $\bar{\mathsf{I}}$ is *prohibited* at state W of scene S.

We only allow fully ground illocutions in cases 1 and 2 above. We use formulae 3–5 above to represent normative positions of agents. We do not "hardwire" deontic notions in our semantics: the predicates above represent deontic modalities but not their relationships, that is, we do not specify how obligations, permissions and prohibitions relate. These are captured with rules, conferring generality on our approach as different deontic relationships can be forged, as we show below.

We differentiate between attempts to utter (*att* formulae) and actual utterances (*utt* formulae). Since we aim at heterogeneous agents whose behaviour we cannot guarantee, we create a “sandbox” where agents can utter whatever they want via *att* formulae. However, not everything agents say may be in accordance with the EI – illegal utterances should be appropriately dealt with, that is, they can be discarded and/or may cause sanctions, depending on the deontic notions we want or need to implement. The *utt* formulae are *confirmations* of *att* formulae.

We show in Fig. 2 a sample institutional state. The utterances show a portion

$$\Delta = \left\{ \begin{array}{l} utt(agora, w_2, inform(ag_4, seller, ag_3, buyer, offer(car, 1200), 10)), \\ utt(agora, w_3, inform(ag_3, buyer, ag_4, seller, buy(car, 1200), 13)), \\ obl(payment, w_4, inform(ag_3, payer, ag_4, payee, pay(Price), T_1)), \\ prh(payment, w_2, ask(ag_3, payer, X, adm, leave, T_2)) \\ oav(ag_3, credit, 3000), oav(car, price, 1200), \\ 1200 \leq Price, Price \leq 1200, 13 < T_1 \end{array} \right\}$$

Fig. 2. Sample Institutional State

of the dialogue between a buyer agent and a seller agent – the seller agent ag_4 offers to sell a car for 1200 to buyer agent ag_3 who accepts the offer. The order among utterances is represented via time stamps (10 and 13 in the constructs above). In our example, agent ag_3 has agreed to buy the car so it is assigned an obligation to pay 1200 to agent ag_4 when the agents move to scene *payment*; agent ag_3 is prohibited from asking the scene administrator *adm* to leave the payment scene. We employ a predicate *oav* (standing for *object-attribute-value*) to store attributes of our state: these concern the credit of agent ag_3 and the price of the car. The constraints define the minimum value for *Price*, and the earliest time T_1 ag_3 is obliged to pay.

4 Institutional Rules

Institutional rules are constructs of the form $\mathsf{LHS} \rightsquigarrow \mathsf{RHS}$. LHS contains references to parts of the current institutional state which, if they hold, will cause the rule to be triggered; RHS depicts how the next institutional state is built:

Def. 4. *An institutional rule, denoted as* R*, is defined as:*

$$\begin{array}{rl} \mathsf{R} ::= & \mathsf{LHS} \rightsquigarrow \mathsf{RHS} \\ \mathsf{LHS} ::= & \mathsf{A} \wedge \mathsf{LHS} \mid \neg \mathsf{LHS} \mid \mathsf{A} \\ \mathsf{RHS} ::= & \mathsf{U} \wedge \mathsf{RHS} \mid \mathsf{U} \\ \mathsf{U} ::= & \oplus \mathsf{A} \mid \ominus \mathsf{B} \end{array}$$

Intuitively the LHS depicts the conditions on the institutional state for the rule to apply. The RHS depicts the updates to the institutional state, yielding the next state of affairs. The updates U add (via $\oplus$) and remove (via $\ominus$) atomic formulae A but constraints C cannot be removed – our semantics works by *refining* an institutional state, always adding constraints. Recall that B are atomic formulae which are not constraints. We provide examples of rules in Section 4.3.

4.1 Semantics of Institutional Rules

We now define the semantics of our institutional rules as relationships between institutional states. However, constraints have a special status when they appear both in rules and in institutional states. We start off defining a means to extract the constraints of an institutional state:

Def. 5. *$constrs(\Delta, \Gamma), \Gamma \subseteq \Delta$, holds iff Γ is the smallest set containing all constraints $\mathsf{C} \in \Delta$.*

We also need to define how constraints are checked for their satisfiability – we do so via relationship *satisfy*, defined below. In the definition below $\lhd, \lhd' \in \{<, \leq\}$ are generic means to refer to constraint operators[1]. For simplicity in dealing with all different cases, we assume our constraints to be in a preferred format: $\mathsf{T} > \mathsf{T}'$ and $\mathsf{T} \geq \mathsf{T}'$ are changed to, respectively, $\mathsf{T}' < \mathsf{T}$ and $\mathsf{T}' \leq \mathsf{T}$.

Def. 6. *$satisfy(\Gamma, \Gamma')$ holds, for two sets of constraints Γ, Γ' iff Γ can be satisfied and Γ' is the smallest set obtained from Γ such that:*

- *if both $(\mathsf{T} \lhd X), (X \lhd' \mathsf{T}') \in \Gamma$ then $(\mathsf{T} \lhd X \lhd' \mathsf{T}') \in \Gamma'$.*
- *if $(X \lhd \mathsf{T}) \in \Gamma$ then $(-\infty < X \lhd \mathsf{T}) \in \Gamma'$.*
- *if $(\mathsf{T} \lhd X) \in \Gamma$ then $(\mathsf{T} \lhd X < \infty) \in \Gamma'$.*

Γ' contains a syntactic variation of the elements in Γ in which the constraints of each variable are *expanded* to be within an interval – two limits, $-\infty, \infty$, represent the lowest and highest value any variable may have. Our work builds on standard technologies for constraint solving – in particular, we have been experimenting with SICStus Prolog [9] constraint satisfaction libraries [10, 11]. We can define *satisfy*/2 via SICStus Prolog built-in `call_residue`/2:

$$satisfy(\{\mathsf{C}_1, \ldots, \mathsf{C}_n\}, \Gamma') \leftarrow \texttt{call\_residue}((\mathsf{C}_1, \ldots, \mathsf{C}_n), \Gamma')$$

Predicate `call_residue`/2 takes as its first parameter a sequence of constraints and, if the constraints are satisfiable, returns in its second parameter a list of partially solved constraints. For instance, using SICStus Prolog prefix "#" to operate the finite domain constraint solver, we query and obtain the following:

```
?- call_residue((X + 50 #< Y,Y #< Z + 20,Y #> Z+5,Z #=< 100,Z #> 30),Γ).
   Γ = [[X]-(X in inf..68),[Y]-(Y in 37..119),[Z]-(Z in 31..100)]
```

The representation `LimInf..LimSup` is a syntactic variation of our expanded constraints. We can thus translate Γ above as $\{-\infty < \mathtt{X} < 68, 37 < \mathtt{Y} < 119, 31 < \mathtt{Z} < 100\}$. Our proposal hinges on the existence of the *satisfy* relationship that can be implemented differently: it is only important that it should return a set of partially solved expanded constraints. The importance of the expanded constraints is that they allow us to precisely define when the constraints on the LHS of the rule hold in the current institutional state, as captured by $\sqsubseteq$:

Def. 7. *$\Gamma_1 \sqsubseteq \Gamma_2$ holds iff $satisfy(\Gamma_1, \Gamma_1')$ and $satisfy(\Gamma_2, \Gamma_2')$ hold and for every constraint $(\bot_1 \lhd X \lhd \top_1)$ in Γ_1', there is a constraint $(\bot_2 \lhd X \lhd \top_2)$ in Γ_2', such that $max(\bot_1, \bot_2) \geq \bot_1$ and $min(\bot_1, \bot_2) \leq \bot_1$, where $\bot_i, \top_i, i = 1, 2$ are arbitrary values.*

[1] The remaining operators $=, \neq$ can be defined in terms of $\leq$, that is, $X = \mathsf{T}$ is $\mathsf{T} \leq X \leq \mathsf{T}$ and $X \neq \mathsf{T}$ is $\neg(\mathsf{T} \leq X \leq \mathsf{T})$.

That is, all variables in Γ_1 must be in Γ_2 (with possibly other variables not in Γ_1) and *i)* the maximum value for these variables in Γ_1, Γ_2 must be greater than or equal to the maximum value of that variable in Γ_1; *ii)* the minimum value for these variables in Γ_1, Γ_2 must be less than or equal to the minimum value of that variable in Γ_1. We make use of this relationship to define that constraints of a rule hold in an institutional state if they further limit the values of the existing constrained variables.

We now proceed to define the semantics of an institutional rule. In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation [7, 12]:

Def. 8. *A substitution σ is a finite, possibly empty set of pairs X_i/T_i, $0 \leq i \leq n$.*

The application of σ to a formula A follows the usual definition [12]:

1. $c \cdot \sigma = c$ for a constant c.
2. $X \cdot \sigma = \mathsf{T} \cdot \sigma$ if $X/\mathsf{T} \in \sigma$; if $X/\mathsf{T} \notin \sigma$ then $X \cdot \sigma = X$.
3. $p^n(\mathsf{T}_0, \ldots, \mathsf{T}_n) \cdot \sigma = p^n(\mathsf{T}_0 \cdot \sigma, \ldots, \mathsf{T}_n \cdot \sigma)$.

We now define when the LHS matches an institutional state:

Def. 9. $\mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma)$ *holds depending on the format of* LHS*:*

1. $\mathbf{s}_l(\Delta, \mathsf{A} \wedge \mathsf{LHS}, \sigma_1 \cup \sigma_2)$ *holds iff* $\mathbf{s}_l(\Delta, \mathsf{A}, \sigma_1)$*,* $\mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma_2)$ *hold.*
2. $\mathbf{s}_l(\Delta, \neg\mathsf{LHS}, \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma)$ *does not hold.*
3. $\mathbf{s}_l(\Delta, \mathsf{B}, \sigma)$ *holds iff* $\mathsf{B} \cdot \sigma \in \Delta$*, constrs*$(\Delta, \Gamma)$ *and satisfy*$(\Gamma \cdot \sigma, \Gamma')$*.*
4. $\mathbf{s}_l(\Delta, \mathsf{C}, \sigma)$ *holds iff constrs*(Δ, Γ) *and* $\{\mathsf{C} \cdot \sigma\} \sqsubseteq \Gamma$*.*

Case 1 depicts how substitutions are combined to provide the semantics for conjunctions in the LHS. Case 2 addresses the negation operator. Case 3 states that an atomic formula (which is not a constraint) holds in Δ if it is a member of Δ and the constraints on the variables of Δ hold under σ. Case 4 deals with a constraint: we apply σ to it (thus reflecting the values of matchings of other atomic formula), then check whether the constraint can be included in the state.

We want our institutional rules to be *exhaustively* applied on the institutional state. 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. 10. $\mathbf{s}_l^*(\Delta, \mathsf{LHS}, \Sigma)$ *holds, iff* $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *is the largest non-empty set such that* $\mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma_i), 1 \leq i \leq n$*, holds.*

We must define the application of a set of substitutions $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ to a term T: this results in a set of substituted terms, $\mathsf{T} \cdot \{\sigma_1, \ldots, \sigma_n\} = \{\mathsf{T} \cdot \sigma_1, \ldots, \mathsf{T} \cdot \sigma_n\}$. We now define the semantics of the RHS of a rule as a mapping between the current institutional state Δ and its successor state Δ':

Def. 11. $\mathbf{s}_r(\Delta, \mathsf{RHS}, \Delta')$ *holds depending on the format of* RHS*:*

1. $\mathbf{s}_r(\Delta, (\mathsf{U} \wedge \mathsf{RHS}), \Delta_1 \cup \Delta_2)$ *holds iff* $\mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1)$ *and* $\mathbf{s}_r(\Delta, \mathsf{RHS}, \Delta_2)$ *hold.*
2. $\mathbf{s}_r(\Delta, \oplus\mathsf{B}, \Delta \cup \{\mathsf{B}\})$ *holds.*
3. $\mathbf{s}_r(\Delta, \ominus\mathsf{B}, \Delta \setminus \{\mathsf{B}\})$ *holds.*
4. $\mathbf{s}_r(\Delta, \oplus\mathsf{C}, \Delta \cup \{\mathsf{C}\})$ *holds iff constrs*(Δ, Γ) *and satisfy*$(\Gamma \cup \{\mathsf{C}\}, \Gamma')$ *hold.*

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 B which do not conform to the syntax of constraints. Case 4 defines how a constraint is added to an institutional state: the new constraint is checked for its satisfaction with constraints $\Gamma \subseteq \Delta$ 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 [13, 14], the values to variables obtained when matching the LHS to the institutional state must be passed on to the RHS. We capture this by associating the RHS with a substitution σ obtained when matching the LHS against Δ via $\mathbf{s}_l$. Def. 11 above should actually be used as $\mathbf{s}_r(\Delta, \mathsf{RHS} \cdot \sigma, \Delta')$, that is, we have a version of the RHS with ground variables whose values originate from the matching of the LHS to Δ. We now define how a rule maps two institutional states:

Def. 12. $\mathbf{s}^*(\Delta, \mathsf{LHS} \rightsquigarrow \mathsf{RHS}, \Delta')$ *holds iff* $\mathbf{s}_l^*(\Delta, \mathsf{LHS}, \{\sigma_1, \ldots, \sigma_n\})$ *and* $\mathbf{s}_r(\Delta, \mathsf{RHS} \cdot \sigma_i, \Delta'), 1 \leq i \leq n$, *hold.*

That is, two institutional states Δ, Δ' are related by $\mathsf{LHS} \rightsquigarrow \mathsf{RHS}$ iff we obtain all different substitutions $\{\sigma_1, \ldots, \sigma_n\}$ that make the LHS match Δ and apply these substitutions to RHS in order to build Δ'. Finally we extend $\mathbf{s}^*$ to handle sets of rules: $\mathbf{s}^*(\Delta, \{\mathsf{R}_1, \ldots, \mathsf{R}_n\}, \Delta')$ holds iff $\mathbf{s}^*(\Delta, \mathsf{R}_i, \Delta'), 1 \leq i \leq n$, hold.

4.2 Implementing Institutional Rules

The semantics above provides a basis for an interpreter for institutional rules, shown 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. Clause 1

1. $\mathbf{s}^*(\Delta, \mathsf{Rs}, \Delta') \leftarrow$
   $\texttt{findall}(\langle \mathsf{RHS}, \Sigma \rangle, (\texttt{member}((\mathsf{LHS} \rightsquigarrow \mathsf{RHS}), \mathsf{Rs}), \mathbf{s}_l^*(\Delta, \mathsf{LHS}, \Sigma)), \mathsf{RHSs}),$
   $\mathbf{s}_r'(\Delta, \mathsf{RHSs}, \Delta')$

2. $\mathbf{s}_l^*(\Delta, \mathsf{LHS}, \Sigma) \leftarrow \texttt{findall}(\sigma, \mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma), \Sigma)$
3. $\mathbf{s}_l(\Delta, (\mathsf{A} \wedge \mathsf{LHS}), \sigma_1 \cup \sigma_2) \leftarrow \mathbf{s}_l(\Delta, \mathsf{A}, \sigma_1), \mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma_2)$
4. $\mathbf{s}_l(\Delta, \neg\mathsf{LHS}, \sigma) \leftarrow \neg\mathbf{s}_l(\Delta, \mathsf{LHS}, \sigma)$
5. $\mathbf{s}_l(\Delta, \mathsf{B}, \sigma) \leftarrow \texttt{member}(\mathsf{B} \cdot \sigma, \Delta), \mathit{constrs}(\Delta, \Gamma), \mathit{satisfy}(\Gamma \cdot \sigma, \Gamma')$
6. $\mathbf{s}_l(\Delta, \mathsf{C}, \sigma) \leftarrow \mathit{constrs}(\Delta, \Gamma), \{\mathsf{C} \cdot \sigma\} \sqsubseteq \Gamma$

7. $\mathbf{s}_r'(\Delta, \mathsf{RHSs}, \Delta') \leftarrow$
   $\texttt{findall}(\Delta'', (\texttt{member}(\langle \mathsf{RHS}, \Sigma \rangle, \mathsf{RHSs}), \texttt{member}(\sigma, \Sigma), \mathbf{s}_r(\Delta, \mathsf{RHS} \cdot \sigma, \Delta'')), \mathit{AllDeltas}),$
   $\mathit{merge}(\mathit{AllDeltas}, \Delta')$
8. $\mathbf{s}_r(\Delta, (\mathsf{U} \wedge \mathsf{RHS}), \Delta_1 \cup \Delta_2) \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1), \mathbf{s}_r(\Delta, \mathsf{RHS}, \Delta_2)$
9. $\mathbf{s}_r(\Delta, \oplus\mathsf{B}, \Delta \cup \{\mathsf{B}\})) \leftarrow$
10. $\mathbf{s}_r(\Delta, \ominus\mathsf{B}, \Delta \setminus \{\mathsf{B}\})) \leftarrow$
11. $\mathbf{s}_r(\Delta, \oplus\mathsf{C}, \Delta \cup \{\mathsf{C}\}) \leftarrow \mathit{constrs}(\Delta, C), \mathit{satisfy}([\mathsf{Constr}|C], C')$

Fig. 3. An Interpreter for Institutional Rules

contains the topmost definition: given a Δ and a set of rules Rs, it shows how we can obtain the next state Δ' by finding (via the built-in `findall` predicate[2]) all those rules in Rs (picked by the `member` built-in) whose LHS holds in Δ (checked

[2] ISO Prolog built-in `findall`/3 obtains all answers to a query (2nd argument), recording the values of the 1st argument as a list stored in the 3rd argument.

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 (represented as individual substitutions σ) that the left-hand side LHS of a rule can be matched in an institutional state Δ – the individual σ's are stored in sets Σ of substitutions, as a result of the `findall`/3 execution. Clauses 3-6 are adaptations of Def. 9.

Clause 7 shows how $\mathbf{s}_r'$ computes the new state from a list RHSs of pairs $\langle \mathsf{RHS}, \Sigma\rangle$ (obtained in the second body goal of clause 1): it picks out (via predicate `member`/2) each individual substitution $\sigma \in \Sigma$ and uses it in RHS to compute via $\mathbf{s}_r$ a partial new institutional state Δ'' which is stored in *AllDeltas*. *AllDeltas* contains a set of partial new institutional states and these are combined together via the *merge*/2 predicate – it joins all the partial states, removing any replicated components. A garbage collection mechanism can be also added to the functionalities of *merge*/2 whereby constraints whose variables are not referred in Δ are discarded. Clauses 8-11 are adaptations of Def. 11.

Our interpreter shows how we deal with constraints in our institutional rules: we could not simply refer to standard rule interpreters [13, 14] since these do not handle constraints. Our combination of Prolog built-ins and abstract definitions provides a precise account of the complexity of the computation, yet it is very close to the mathematical definitions.

4.3 Sample Institutional Rules

In this section we give examples of institutional rules and explain the computational behaviours they capture. These examples illustrate what can be achieved with our proposed formalism.

To account for the passing of time, we shall simulate a clock using our institutional rules. Our clock is used by the governor and institutional agents when they are writing terms onto the tuple space (see discussion in Section 5 below). We shall represent our clock as the term $now(T)$ where T is a natural number. This term can either be provided in the initial state or a "bootstrapping" rule can be supplied as $\neg now(T) \rightsquigarrow \oplus now(1)$, that is, if $now(T)$ is not present in the state, the rule will add $now(1)$ to the next state. Similar rules can be used whenever new terms need to be added to the state, but once added they only need to be updated.

We can simulate the passing of time in various ways. A *reactive* approach whereby an *event* triggers a rule to update the *now* term is captured as:

$$(now(T) \wedge att(S, W, P(A_1, R_1, A_2, R_2, M, T)) \rightsquigarrow (\ominus now(T) \wedge \oplus now(T+1)))$$

That is, if an event (an attempt to utter something) related to the current time step has happened then the clock is updated. It is important to notice that if there is more than one utterance, the exhaustive application of the rule above will carry out the same update for each utterance. Although this might be unnecessary or inefficient, it will not cause multiple $now/1$ formulae to be inserted in the next institutional state, as the same unification for T is used in all updates, rendering the same $T + 1$.

If external agents fail to provide their messages (via their governor agents, as explained below), this can be then neatly captured by a "dummy" message. Governor agents can be defined to wait a predefined amount of time and then write out in the institutional state that a timeout took place – this can be represented by $utt(S, W, P(A, R, adm, admin, timeout, T))$, stating that agent A failed to say what it should have said.

We can also define relationships among permissions, prohibitions and obligations via our institutional rules. Such relationships should capture the pragmatics of normative aspects – what exactly these concepts mean in terms of agents' behaviour. We start by looking at those illocutions that external agents attempted to utter, *i.e.*, $att(S, W, I)$:

$$(att(S, W, I) \wedge per(S, W, I)) \rightsquigarrow (\ominus att(S, W, I) \wedge \oplus utt(S, W, I))$$

That is, permitted attempts become utterances – any constraints associated with S, W and I should hold for the left-hand side to match the current state.

Attempts and prohibitions can be related together by the schematic rule

$$(att(S, W, I) \wedge prh(S, W, I)) \rightsquigarrow Sanction$$

Where *Sanction* stands for sanctions on the agents who tried to utter a prohibited illocution. If we represent the credit of agents as $oav(Ag, credit, V)$, we can apply a 10% fine on those agents who attempt to utter a prohibited illocution:

$$\left(\begin{array}{c} att(S, W, P(A_1, R_1, A_2, R_2, M, T)) \wedge \\ prh(S, W, P(A_1, R_1, A_2, R_2, M, T)) \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \ominus oav(A_1, credit, C) \wedge \\ \oplus oav(A_1, credit, C - C/10) \wedge \\ \ominus att(S, W, P(A_1, R_1, A_2, R_2, M, T)) \end{array}\right)$$

Another way of relating attempts, permissions and prohibitions is when a permission granted in general (*e.g.*, to all agents or to all agents adopting a role) is revoked for a particular agent (*e.g.*, due to a sanction). We can ensure that a permission has not been revoked via the rule

$$(att(S, W, I) \wedge per(S, W, I) \wedge \neg prh(S, W, I)) \rightsquigarrow (\ominus att(S, W, I) \wedge \oplus utt(S, W, I))$$

That is, only permitted attempts which are not prohibited become utterances.

We can also capture other relationships among deontic modalities. For instance, the rule below states that all obligations are also permissions:

$$obl(S, W, I) \rightsquigarrow \oplus per(S, W, I)$$

Such a rule would add to the institutional state a permission for every obligation. Another relationship we can forge concerns how to cope with the situation when an illocution is an obligation and a prohibition – this may occur when an obligation assigned to agents in general (or to any agents playing a role) is revoked for individual agents (for instance, due to a sanction). In this case, we can choose to ignore/override either the obligation or the prohibition. For instance, the rule below overrides the obligation and ignores the attempt to fulfil the obligation:

$$(att(S, W, I) \wedge obl(S, W, I) \wedge prh(S, W, I)) \rightsquigarrow \ominus att(S, W, I)$$

The rule below ignores the prohibition and transforms an attempt to utter the illocution I into its utterance:

$$(att(S, W, I) \wedge obl(S, W, I) \wedge prh(S, W, I)) \rightsquigarrow (\ominus att(S, W, I) \wedge \oplus utt(S, W, I))$$

A third possibility is to raise an exception via a term which can then be dealt with at the institutional level. The following rule could be used for this purpose:

$$(att(S, W, I) \wedge obl(S, W, I) \wedge prh(S, W, I)) \rightsquigarrow \oplus exception(S, W, I)$$

We do not want to be prescriptive in our discussion and we are aware that the sample rules we present can be given alternative formulations. Furthermore, we notice that when designing institutional rules, it is essential to consider the *combined* effect of the whole set of rules over the institutional states – these should be engineered in tandem.

The rules above provide a sample of the expressiveness and precision of our institutional rules. As with all other formalisms to represent computations, it is difficult to account for the *pragmatics* of institutional rules. Ideally, we should provide a list of *programming techniques* engineers are likely to need in their day-to-day activities, but this lies outside the scope of this paper.

5 An Architecture for Norm-Aware Agent Societies

We now elaborate on the distributed architecture which fully defines our normative (or social) layer to EIs. We refer back to **Fig 1**, the initial diagram describing our proposal. We show in the centre of the diagram a tuple space [6] – this is a blackboard system with accompanying operations to manage its entries. Our agents, depicted as a rectangle (labelled *IAg*), circles (labelled *GAg*) and hexagons (labelled *EAg*) interact (directly or indirectly) with the tuple space, reading and deleting entries from it as well as well as writing entries onto it. We explain the functionalities of each of our agents below. The institutional states $\Delta_0, \Delta_1, \ldots$ are recorded in the tuple space; we propose a means to represent institutional states with a view to maximising asynchronous aspects (*i.e.*, agents should be allowed to access the tuple space asynchronously) and minimising housekeeping (*i.e.*, not having to move information around).

The topmost rectangle in Fig. 1 depicts our *institutional agent IAg*, responsible for updating the institutional state, applying $\mathbf{s}^*$. The circles below the tuple space represent the *governor agents GAg*s, responsible for following the EI "chaperoning" the *external agents EAg*s. The external agents are arbitrary heterogeneous software or human agents that actually enact an EI to ensure that they conform to the required behaviour, each external agent is provided with a governor agent with which it communicates to take part in the EI. Governor agents ensure that external agents fulfil all their social duties during the enactment of an EI. In our diagram, we show the access to the tuple space as black block arrows; communication among agents are the white block arrows.

We want to make the remaining discussion as concrete as possible so as to enable others to assess, reuse and/or adapt our proposal. We shall make use of SICStus Prolog [9] Linda Tuple Spaces [6] library in our discussion. A Linda tuple space is basically a shared knowledge base in which terms (also called tuples or entries) can be asserted and retracted asynchronously by a number of distributed processes. The Linda library offers basic operations to read a tuple from the space (predicates `rd/1` and its non-blocking version `rd_noblock/1`), to remove a tuple from the space (predicates `in/1` and its non-blocking version `in_noblock/1`), and to write a tuple onto the space (predicate `out/1`). Messages are exchanged among the governor agents by writing them onto and reading them from the tuple space; governor agents and their external agents, however, communicate via exclusive point-to-point communication channels.

In our proposal some synchronisation is necessary: the utterances $utt(s, w, \mathsf{I})$ will be written by the governor agents and the external agents must provide the actual values for the variables of the messages. However, governor agents must stop writing illocutions onto the space so that the institutional agent can update the institutional state. We have implemented this via the term `current_state(N)` (`N` being an integer) that works as a flag: if this term is present on the tuple space then governor agents may write their utterances onto the space; if it is not there, then they have to wait until the term appears. The institutional agent is responsible for removing the flag and writing it back, at appropriate times.

We show in Fig. 4 a Prolog implementation for the institutional agent *IAg*. It bootstraps the architecture by creating an initial value 0 for the current state (lines 2-3); the initial institutional state is empty. In line 3 the institutional agent obtains via `time_step`/1 a value `T`, an attribute of the EI enactment setting up the frequency new institutional states should be computed.

The *IAg* agent then enters a loop (lines 5-14) where it initially (line 6) sleeps for `T` milliseconds – this guarantees that the frequency of the updates will be respected. *IAg* then checks via `no_one_updating`/0 (line 7) that there are no governor agents currently updating the institutional state with their utterances – `no_one_updating`/0 succeeds if there are no `updating`/2 tuples in the space. Such tuples are written by the governor agents to inform the institutional agent it has to wait until their utterances are written onto the space.

When agent *IAg* is sure there are no more governor agents updating the tuple space then it removes the `current_state`/1 tuple (line 8) thus preventing any governor agent from trying to update the tuple space (the governor agent checks in line 7 of **Fig. 5** if such entry exists – if it does not, then the flow of execution is blocked on that line). Agent *IAg* then obtains via predicate `get_state`/2 all those tuples pertaining to the current institutional state `N` and stores them in `Delta`; the institutional rules are obtained in line 10 – they are also stored in the tuple space so that any of the agents can examine them. In line 11 `Delta` and `Rules` are used to obtain the next institutional state `NewDelta` via predicate $\mathtt{s}^*$/2 (cf. Def. 12 and its implementation in Fig 3). In line 12 the new institutional state `NewDelta` is written onto the tuple space, then the tuple recording the identification of the current state is written onto the space (line 14) for the next update. Finally, in line 15 the agent loops[3].

```
 1 main:-
 2   out(current_state(0)),
 3   time_step(T),
 4   loop(T).

 5 loop(T):-
 6   sleep(T),
 7   no_one_updating,
 8   in(current_state(N)),
 9   get_state(N,Delta),
10   inst_rules(Rules),
11   s*(Delta,Rules,NewDelta),
12   write_onto_space(NewDelta),
13   NewN is N + 1,
14   out(current_state(N)),
15   loop(T).
```

Fig. 4: Institutional Agent

Distinct threads will execute the code for the governor agents *GAg* shown in Fig. 5. Each of them will connect to an external agent via predicate

[3] For simplicity we did not show the termination conditions for the loops of the institutional and governor agents. These conditions are prescribed by the EI specification and should appear as a clause preceding the loop clauses of Figs. 4 and 5.

`connect_ext_ag/1` and obtain its identification `Ag`, then find out (line 3) about the EI's root scene (where all agents must initially report to [5]) and that scene's initial state (line 4) – we adopt here the representation for EIs proposed in [8]. In line 5 the governor agent makes the initial call to `loop/1`: the `Role` variable is not yet instantiated at that point, as a role is assigned to the agent when it joins the EI. The governor agents then will loop through lines 6-15, initially checking in line 7 if they are allowed to update the current institutional state, adding their utterances. Only if the `current_state/1` tuple is on the space then does the flow of execution of the governor agent move to line 8, where it obtains the identifier `Ag` from the control list `Ctl`; in line 9 a tuple `updating/2` is written out onto the space. This tuple informs the institutional agent that there are governors updating the space and hence it should wait to update the institutional state. In line 10 the governor agent reads all those tuples pertaining to the current institutional state. In line 11 the governor agent collects all those actions `send/1` and `receive/1` in the EI specification which are associated with its current control `[Ag,Sc,St,Role]`. In line 12, the governor agent interacts with the external agent and, taking into account all constraints associated with `Ag`, obtains an action `Act` that is performed in line 14 (*i.e.*, a message is sent or received). In line 14 the agent removes the `updating/2` tuple and in line 15 the agent starts another loop.

```
 1 main:-
 2  connect_ext_ag(Ag),
 3  root_scene(Sc),
 4  initial_state(Sc,St),
 5  loop([Ag,Sc,St,Role]).

 6 loop(Ctl):-
 7  rd(current_state(N)),
 8  Ctl = [Ag|_],
 9  out(updating(Ag,N)),
10  get_state(N,Delta),
11  findall([A,NC],(p(Ctl):-A,p(NC)),ANCs),
12  social_analysis(ANCs,Delta,Act,NewCtl),
13  perform(Act),
14  in(updating(Id,N)),
15  loop(NewCtl).
```

Fig. 5: Governor Agent

Although we represented the institutional state as boxes in **Fig 1** they are not stored as one single tuple containing Δ. If this were the case, then the governors would have to take turns to update the institutional state. We have used instead a representation for the institutional state that allows the governors to update the space asynchronously. Each element of Δ is represented by a tuple of the form `t(N,Type,Elem)` where `N` is the identification of the institutional state, `Type` is the description of the component (*i.e.*, either a `rule`, an `atf`, or a `constr`) and `Elem` the actual element.

Using this representation, we can easily obtain all those tuples in the space that belong to the current institutional state. Predicate `get_state/2` is thus:

```
get_state(N,Delta):- bagof_rd_noblock(t(N,T,E),t(N,T,E),Delta).
```

That is, the Linda built-in [9] `bagof_rd_noblock/3` (it works like the `findall/3` predicate) finds all those tuples belonging to institutional state `N` and stores them in `Delta`.

5.1 Norm-Aware Governor Agents

We can claim our resulting society of agents is endowed with norm-awareness because their behaviour is regulated by the governor agents depicted above. The social awareness of the governor agent, in its turn, stems from two features: *i)* its access to the institutional state where obligations, prohibitions and permissions are recorded (as well as constraints on the values of their variables); *ii)* its access to the set of possible actions prescribed in the protocol. With this information, we can define various alternative ways in which governor agents, in collaboration with their respective external agents, can decide on which action to carry out.

We can define predicate `social_analysis(ANCs,Delta,Act,NewCtr)` in line 12 of Fig. 5 in different ways – this predicate should ensure that an action `Act` (sending or receiving a message) with its respective next control state `NewCtr` (*i.e.*, the list `[Ag,Sc,NewSt,Role]`) is chosen from the list of options `ANCs`, taking into account the current institutional state `Delta`. This predicate must also capture the interactions between governor and external agents as, together, they choose and customise a message to be sent.

We show in Fig. 6 a definition for predicate `social_analysis`/4. Its first subgoal removes from the list `ANCs` all those utterances that are prohibited from being sent, obtaining the list `ANCsWOPrhs`. The second subgoal ensures that obligations are given adequate priority: the list `ANCsWOPrhs` is further refined to get the obligations among the actions and store them in list `ANCsObls` – if there are no obligations, then `ANCsWOPrhs` is the same as `ANCsObls`. Finally, in the third subgoal, an action is chosen from `ANCsObls` and customised in collaboration with the external agent. This definition is a rather "draconian" one in which external agents are never allowed even to attempt to utter a prohibited illocution; other definitions could be supplied instead.

```
social_analysis(ANCs,Delta,Act,NewCtr):-
  remove_prhs(ANCs,Delta,ANCsWOPrhs),
  select_obls(ANCsWOPrhs,Delta,ANCsObls),
  choose_customise(ANCsObls,Delta,Act,NewCtr).
```

Fig. 6: Definition of Social Analysis

We use a "flat" structure to represent atomic formulae. For instance,

$$utt(agora, w_2, inform(ag_4, seller, ag_3, buyer, offer(car, 1200), 10))$$

is represented as

t(N,atf,[utt,agora,w_2,[inform,ag_4,seller,ag_3,buyer,offer(car,1200),10]])

Governor agents are able to answer queries by their external agents such as "what are my obligations at this point?", encoded as:

```
findall([S,W,[I,Id|R]],member(t(N,atf,[obl,S,W,[I,Id|R]]),Delta),MyObls)
```

These interactions enrich the definition of predicate `choose_customise`/4 above.

6 Related Work

Apart from classical studies on law, research on norms and agents has been addressed by two different disciplines: sociology and philosophy. On the one hand, socially oriented contributions highlight the importance of norms in agent behaviour (*e.g.*, [15–17]) or analyse the emergence of norms in multi-agent systems (*e.g.*, [18, 19]). On the other hand, logic-oriented contributions focus on the deontic logics required to model normative modalities along with their paradoxes (*e.g.*, [20–22]). The last few years, however, have seen significant work on norms in multi-agent systems, and norm formalisation has emerged as an important research topic in the literature [1, 23–25].

Vázquez-Salceda *et al.* [24, 26] propose the use of a deontic logic with deadline operators. In their approach, they distinguish norm conditions from violation conditions. This is not necessary in our approach since both types of conditions can be represented in the LHS of our rules. Their model of norm also separates sanctions and repairs (*i.e.*, actions to be done to restore the system to a valid state); these can be expressed in the RHS of our rules without having to differentiate them from other normative aspects of our states. Our approach has two advantages over [24, 26]: *i)* we provide an implementation for our rules; and *ii)* we offer a more expressive language with constraints over norms.

Fornara *et al.* [25] propose the use of norms partially written in Object Constraint Language (OCL). Their commitments are used to represent all normative modalities; of special interest is how they deal with permissions: they stand for the absence of commitments. This feature may jeopardise the safety of the system since it is less risky to only permit a set of safe actions thus forbidding other actions by default. Although this feature can reduce the amount of permitted actions, it allows unexpected actions to be carried out. Their *within*, *on* and *if* clauses can be encoded as LHS of our rules as they can all be seen as conditions when dealing with norms. Similarly, "*foreach in*" and "*do*" clauses can be encoded as RHS of our rules since they are the actions to be applied to a set of agents.

López y López *et al.* [27] present a model of normative multi-agent system specified in the Z language. Their proposal is quite general since the normative goals of a norm do not have a limiting syntax as is the case with the rules of Fornara *et al.* [25]. However, their model assumes that all participating agents have a homogeneous, predetermined architecture. No agent architecture is imposed on the participating agents in our approach, thus allowing for heterogeneity.

Artikis *et al.* [28] use event calculus for the specification of protocols. Obligations, permissions, empowerments, capabilities and sanctions are formalised by means of fluents (*i.e.*, predicates that may change with time). Prohibitions are not formalised in [28] as a fluent since they assume that every action not permitted is forbidden by default. Although event calculus models time, their deontic fluents 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.

Michael *et al.* [29] propose a formal scripting language to model the essential semantics, namely, rights and obligations, of market mechanisms. They also formalise a theory to create, destroy and modify objects that either belong to someone or can be shared by others. Their proposal is suitable to model and implement market mechanisms. However, it is not as expressive as other proposals: for instance, it cannot model obligations with a deadline.

7 Conclusions, Discussion and Future Work

We have proposed a distributed architecture to provide MASs with an explicit social layer: the *institutional states* store information on the execution of the MAS as well as the normative positions of its agents – their obligations, prohibitions and permissions. The institutional states capture the dynamics of the execution and are managed via *institutional rules*: these are a kind of production system depicting how the states are updated when certain situations arise.

An important contribution of this work concerns the rule-based language to explicitly manage normative positions of agents. We achieve greater flexibility, expressiveness and precision by allowing *constraints* to be part of our rules – such constraints associate further restrictions with permissions, prohibitions and obligations. Our language is general-purpose, allowing various kinds of deontic notions to be captured.

The institutional states and rules are put to use within a distributed architecture, supported by a team of administrative agents implemented as Prolog programs sharing a tuple space. We propose means to store the institutional state that allows maximum distributed access. The "norm-awareness" of our proposal stems from the fact that the governor agents, part of our team of administrative agents, can regulate the behaviour of external agents taking part in the MAS execution. The regulation takes into account the normative position of individual external agents stored in the institutional state. We provide a detailed implementation of governor agents that hinges on the notion of social analysis: this is a decision procedure which can be defined differently, for distinct scenarios and solutions.

We would like to investigate the verification of norms (along the lines of our work in [30]) expressed in our rule language, with a view to detecting, for instance, obligations that cannot be fulfilled, prohibitions that prevent progress, inconsistencies (*i.e.*, when an illocution is simultaneously permitted and prohibited) and so on. We also want to provide engineers with means to analyse their rules, so that they can, for instance, assess the "social burden" associated with individual agents and whether any particular agent has too important a role in the progress of an electronic institution.

If the verification and analysis are done *during* the design, that is, as the rules are prepared, then this could prevent problems from being propagated to latter parts of the MAS development. We are currently working on tools to help engineers prepare and analyse their rules; these are norm editors that will support the design of norm-oriented electronic institutions.

References

1. Dignum, F.: Autonomous Agents with Norms. A. I. & Law **7** (1999) 69–79
2. López y López, F., Luck, M., d'Inverno, M.: Constraining Autonomy Through Norms. In: Procs. AAMAS 2002, ACM Press (2002)
3. Verhagen, H.: Norm Autonomous Agents. PhD thesis, Stockholm University (2000)
4. Sergot, M.: A Computational Theory of Normative Positions. ACM Trans. Comput. Logic **2** (2001)
5. Esteva, M.: Electronic Institutions: from Specification to Development. PhD thesis, Universitat Politècnica de Catalunya (UPC) (2003) IIIA monography Vol. 19.
6. Carriero, N., Gelernter, D.: Linda in Context. Comm. of the ACM **32** (1989)
7. Apt, K.R.: From Logic Programming to Prolog. Prentice-Hall, U.K. (1997)
8. Vasconcelos, W.W., Robertson, D., Sierra, C., Esteva, M., Sabater, J., Wooldridge, M.: Rapid Prototyping of Large Multi-Agent Systems through Logic Programming. Annals of Mathematics and Artificial Intelligence **41** (2004) 135–169

9. Swedish Institute of Computer Science: SICStus Prolog. (2005) `http://www.sics.se/isl/sicstuswww/site/index.html`, viewed on 10 Feb 2005 at 18.16 GMT.
10. Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The Semantics of Constraint Logic Programs. Journal of Logic Programming **37** (1998) 1–46
11. Holzbaur, C.: ÖFAI clp(q,r) Manual, Edition 1.3.3. TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, Austria (1995)
12. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, U.S.A. (1990)
13. Kramer, B., Mylopoulos, J.: Knowledge Representation. In Shapiro, S.C., ed.: Encyclopedia of Artificial Intelligence. Volume 1. John Wiley & Sons (1992)
14. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. 2 edn. Prentice Hall, Inc., U.S.A. (2003)
15. Conte, R., Castelfranchi, C.: Understanding the Functions of Norms in Social Groups through Simulation. In: Artificial Societies. The Computer Simulation of Social Life, UCL Press (1995)
16. Conte, R., Castelfranchi, C.: Norms as Mental Objects: From Normative Beliefs to Normative Goals. In: Procs. of MAAMAW'93, Neuchatel, Switzerland (1993)
17. Tuomela, R., Bonnevier-Tuomela, M.: Norms and Agreement. European Journal of Law, Philosophy and Computer Science **5** (1995) 41–46
18. Walker, A., Wooldridge, M.: Understanding the emergence of conventions in multi-agent systems. In: Procs. ICMAS 2005, San Francisco, USA (2005)
19. Shoham, Y., Tennenholtz, M.: On Social Laws for Artificial Agent Societies: Off-line Design. Artificial Intelligence **73** (1995) 231–252
20. von Wright, G.: Norm and Action: A Logical Inquiry. Routledge and Kegan Paul, London (1963)
21. Alchourron, C., Bulygin, E.: The Expressive Conception of Norms. In Hilpinen, R., ed.: New Studies in Deontic Logics, London, D. Reidel (1981) 95–124
22. Lomuscio, A., Nute, D., eds.: Procs. of DEON 2004. Volume 3065 of LNAI. Springer Verlag (2004)
23. Boella, G., van der Torre, L.: Permission and Obligations in Hierarchical Normative Systems. In: Procs. ICAIL 2003, ACM Press (2003)
24. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing Norms in Multi-agent Systems. Volume 3187 of LNAI., Springer-Verlag (2004)
25. Fornara, N., Viganò, F., Colombetti, M.: A Communicative Act Library in the Context of Artificial Institutions. In: Procs. EUMAS. (2004)
26. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Norms in Multiagent Systems: Some Implementation Guidelines. In: Procs. EUMAS. (2004)
27. López y López, F., Luck, M.: A Model of Normative Multi-Agent Systems and Dynamic Relationships. Volume 2934 of LNAI., Springer-Verlag (2004)
28. Artikis, A., Kamara, L., Pitt, J., Sergot, M.: A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. Volume 3476 of LNAI. Springer-Verlag (2004)
29. Michael, L., Parkes, D.C., Pfeffer, A.: Specifying and monitoring market mechanisms using rights and obligations. In: Proc. AMEC VI. (2004)
30. Vasconcelos, W.W.: Norm Verification and Analysis of Electronic Institutions. Volume 3476 of LNAI. Springer-Verlag (2004)