Implementing Norms in Electronic Institutions

A. Garcia-Camino, P. Noriega, J. A. Rodríguez-Aguilar
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC
Campus de la UAB
08193 Bellaterra, Barcelona, Spain
{andres,jar,pablo}@iiia.csic.es

ABSTRACT
Ideally, open multi-agent systems (MAS) involve heterogeneous and autonomous agents whose interactions ought to conform to some shared conventions. The challenge is how to express and enforce such conditions so that truly autonomous agents can adscribe to them. One way of addressing this issue is to look at MAS as environments regulated by some sort of normative framework. There have been significant contributions to the formal aspects of such normative frameworks, but there are few proposals that have made them operational. In this paper a possible step towards closing that gap is suggested. A normative language is introduced which is expressive enough to represent the familiar types of MAS-inspired normative frameworks; its implementation in JESS is also shown. This proposal is aimed at adding flexibility and generality to electronic institutions by extending their deontic components through richer types of norms that can still be enforced on-line.

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,Implementation,Expert system

1. INTRODUCTION
Multi-agent systems (MAS) have emerged as a promising approach for creating agile information systems suited for addressing problems that have multiple problem-solving entities [12]. MAS are considered *open systems* [11] when the following essential characteristics prevail [19]:

Hetereogeneity. Open MAS must be capable of acommodating heterogeneous agents, i.e. agents possibly developed in different languages, by different parties, with different purposes and preferences.

Reliability. "Open systems must be reliable. They must be designed so that failures of individual components can be accomodated by operating components while the failed components are repaired or replaced."[11]

Accountability and legitimacy. Since agents may possibly exhibit either deviating or fraudulent behaviour, their actions must be monitored to detect and prevent dysfunctional behaviours which may jeopardise the overall functioning of the system.

Societal change. Agent societies are not static; they may evolve over time by altering, eliminating or incorporating rules. Hence the need of demanding flexible societies capable of accomodating future changes.

In this work we assume that open MAS put together heterogeneous, self-interested agents whose actions might deviate from expected behaviour. Moreover, their uncontrolled actions may be harmful to other agents and even to the multi-agent system, leading to unwanted states. In this setting, there is a pending, fundamental issue: the constitution of safe environments that guarantee the constraining of agents' behaviours without restricting their essential characteristic: autonomy. Along this direction, normative systems allow to ensure that agents' behaviours will never bring about unwanted states by specifying the set of illegal actions to be forbidden under particular conditions.

We differentiate three main research lines dealing with normative systems: theoretical models of norms, formal specification of norms and computational models. Current work on theoretical models focuses mainly on the formalization of normative systems with deontic logics [14]. For instance, Dignum et al. propose a variation of deontic logic that includes conditional and temporal aspects [3][5]. These approaches are fundamentally theoretical and have no current implementation.

As to formal specifications, a remarkable example of normative MAS model is the extension of the SMART agent specification framework by López y López et al. [16] [15].

They tackle norm reasoning from an agent-centered perspective, defining different types of agents depending on their strategy to comply with norms (for instance, a *greedy* agent would choose to comply with the set of norms that maximize its benefits). A different perspective is taken in [20] where the major concern is to offer an general definition of norm (which integrates conditional and temporal aspects) from an organizational point of view. Finally, as for computational models, current normative frameworks are either domain specific or their normative component is not expressive and flexible enough. An example of the former is the implementation realized by Michael et al. [17], which permits the specification of market mechanims by the definition of rights, permissions and obligations. An example of the latter is AMELI [7], which makes operational the notion of Electronic Institution (EI for short), guaranteeing the preservation of a legal state of the environment. However, its normative component seriously restricts agents' behaviours by imposing actions when norms are activated. Another recent computational model is based on the use of event calculus for the formalization of norm-governed computational systems [2][1]. Obligations, permissions and prohibitions are expressed as changing predicates called *fluents*. Although some examples have been implemented in *Prolog*, this formalization focuses on norms triggered by actions, it does not include the formalization of norms triggered by temporal issues.

Although there are new emerging approaches, there is still a gap between theoretical proposals and computational models. On the one hand, there are worthy theoretical models and specifications with no implementation [14] [3] [5] [15] [20]. On the other hand, there are robust frameworks which lack the degree of expressiveness and flexibility needed for a normative system [17] [7].

The objective of this paper is to try to fill this gap by adapting a formal approach [20] to enrich an existing framework [7] . More precisely, we have extended the normative language of an EI to increase its expressiveness and flexibility. We have also proposed the use of Jess [13] for the implementation of the norm engine which mantains the normative state of an institution, i.e. the permissions, prohibitions and obligations that hold in the current state of execution. Our implementation has been carried out by translating the norms specified in our normative language into Jess rules. At run-time our norm engine can be updated with new utterances and queried about permissions, prohibitions or pending obligations.

It is worth remarking that we consider autonomous norm compliance from an institutional point of view. That is, we do not care how an agent decides which norms to comply with, but instead we define the norms and the sanctions to be applied when the violation of norms occurs as part of the institution. With this approach we allow agents to reason about norm compliance while the choice and implementation of the agents' architecture is left to the agent developers.

The paper is organised as follows. In section 2 we summarise the notion of EI. Next, we define the normative language in section 3 and show its implementation in section 4. We draw some conclusions and outline our future work in section 5.

2. ELECTRONIC INSTITUTIONS

Electronic institutions, as we consider them [6] [19] [18], shape agent environments that restrict the behaviour of agents to ensure that agents interact in safe conditions. EIs constraint agent behaviour by defining the valid sequences of dialogical interactions that agents can hold to attain their goals. We differentiate two types of norms in EIs: protocol-based and rule-based. Protocol-based norms are defined by a group of scenes, a performative structure, and a dialogical framework that establish the permitted actions at each instant of time taking into account the past actions of agents. Rule-based norms are defined by a certain type of first-order formulae that establish a dependency relation between actions. Some actions under certain conditions fire normative rules which produce new commitments, establishing new pending obligations (actions to be carried out by agents).

2.1 Protocol-based norms

The dialogical framework defines all the conventions required to make the interaction between two or more agents possible. Moreover, it defines what the participant roles within the EI and the relationships among them will be. We take interactions to be a sequence of speech acts between two or more parties. Formally, we express speech acts as illocutionary formulae of the form: $\iota(speaker, hearer, \ , t)$. The speech acts that we use start with an illocutionary particle (declare, request, promise) that a *speaker* addresses to a *hearer*, at time t, whose content is expressed in some object language whose vocabulary stems from an EI's ontology.

A *dialogical framework* encompasses all the illocutions available to the agents in a given institution. Formally,

Def. 1. *A dialogical framework is a tuple* $DF = \langle O, L_O, P, R, R_S \rangle$ *where* O *stands for an ontology (vocabulary);* L_O *stands for a content language to express the information exchanged between agents using ontology* O*;* P *is the set of illocutionary particles;* R *is the set of roles;* R_S *is the set of relationships over roles.*

For each activity in an institution, interactions between agents are articulated through agent group meetings, which we call *scenes*. A scene is a role-based multi-agent protocol specification. A scene defines the valid sequences of interactions among agents enacting different roles. It is defined as a directed graph where each node stands for scene state and each edge connecting two states is labelled by an illocution scheme. An illocution scheme is an illocutionary formula with some unbound variables. At run-time, agents playing different roles make a scene evolve by uttering illocutions that match the illocution schemes connecting states. Each scene mantains the context of the interaction, that is how the dialogue is evolving, i.e. which have been the uttered illocutions and how the illocution schemes have been instantiated.

Def. 2. *A scene is a tuple* $S = \langle s, R, DF, W, w_0, W_f, \Theta, \lambda, min, Max \rangle$ *where* s *is the scene identifier;* R *is the set of scene roles;* DF *is a dialogical framework;* W *is the set of scene states;* $w_0 \in W$ *is the initial state;* $W_f \subseteq W$ *is the set of final states;* $\theta \subseteq W \times W$ *is a set of directed edges;* $\lambda : \theta \longrightarrow \mathcal{L}_{DF}$ *is a labelling function, which maps each edge to an illocution scheme in the pattern language of the*

DF dialogical framework; $min, Max; \mathbb{R} \longrightarrow \mathbb{N}$ $min(r)$ and $Max(r)$ are, respectively, the minimum and the maximum number of agents that must and can play each role $r \in R$.

The activities in an EI are the composition of multiple, distinct, possibly concurrent, dialogical activities, each one involving different groups of agents playing different roles. A performative structure can be seen as a network of scenes, whose connections are mediated by transitions (a special type of scene), and determines the role-flow policy among the different scenes by showing how agents, depending on their roles and prevailing commitments, may get into different scenes, and showing when new scenes will be started. The performative structure defines the possible orders of execution of the interaction protocols (*scenes*). It also allows agent synchronization, and scene interleaved execution.

DEF. 3. *A performative structure is a tuple $PS = \langle S, T, s_0, s_\Omega, E, f_L, f_T, f_E^O, \mu\rangle$ where S is a finite, non-empty set of scenes; T is a finite, non-empty set of transitions; $s_0 \in S$ is the initial scene; $s_\Omega \in S$ is the final scene; $E = E^I \cup E^O$ is a set of edge identifiers where $E^I \subseteq S \times T$ is a set of edges from scenes to transitions and $E^O \subseteq T \times S$ is a set of edges from transitions to scenes; $f_L : E \longrightarrow DNF_{2^{V_A \times R}}$ maps each edge to a disjunctive normal form of pairs of agent variable and role identifier representing the edge label; $f_T : T \longrightarrow \mathcal{T}$ maps each transition to its type; $f_E^O : E^O \longrightarrow \mathcal{E}$ maps each edge to its type; $\mu : S \longrightarrow \{0, 1\}$ sets if a scene can be multiply instantiated at execution time;*

The institutional state consists of the list of scene executions (described by their participanting agents and interaction context) along with the participating agents' state (represented by their observable attributes).

2.2 Normative rules

As mentioned above, actions within an electronic institution are speech acts. Those speech acts that are made in accordance with the performative structure of an institution may create obligations or commitments on participants. Commitment fulfillment needs to be warranted by the institution. We make such intended effects of commitments explicit through what we have called *normative rules.*

DEF. 4. *Normative rules are first-order formulae of the form*

$$(\bigwedge_{i=1}^{n} uttered(s_i, w_k, i_{l_i}) \wedge \bigwedge_{j=0}^{m} e_j) \rightarrow$$
$$(\bigwedge_{i=1}^{n'} uttered(s'_i, w'_k, i'_{l_i}) \wedge \bigwedge_{j=0}^{m'} e'_j)$$

satisfying that $t \ \geq t_i, 1 \leq i \leq n, t' \ \leq t'_j, 1 \leq j \leq n', t \ < t'$ where s_i, s'_i are scene identifiers, w_k, w'_k are states of s_i and s'_i respectively; i_{l_i}, i'_{l_i} are illocutions schemes l_i of s_i and s'_i respectively; and $t \ , t_i$ are the time stamps of, respectively, i_i and i_j, t being the greatest value of time stamp on the left-hand side illocutions (that is, the time stamp of the latest illocution) and t' the lowest value of the time stamp on the right-hand side illocutions (that is, the time stamp of the earliest illocution); e_j, e'_j are boolean expressions over variables from the illocution schemes i_{l_i} and i'_{l_i}, respectively.

The intuitive meaning of normative rules is that they create obligations in the sense that if grounded illocutions matching $i_{l_1}, \ldots, i_{l_n}$ are uttered in the corresponding scene states, and expressions $e_1, \ldots, e_m$ are satisfied, then, grounded illocutions matching $i'_{l_1}, \ldots, i'_{l_{n'}}$ satisfying the expressions $e'_1, \ldots, e'_{m'}$ must be uttered in the corresponding scene states.

3. NORMATIVE LANGUAGE

In the definition stated above normative rules are strict in the sense that entailed actions are forced upon the agents at given scenes and states. However, it is convenient to have, also, less strict normative rules where action execution might be dependent on temporal constraints or the occurrence of events.

Although normative rules are an efficient method to formalize strict obligations, there is also the need for specifying permissions, prohibitions and obligations with conditional and temporal constraints. As an exercise in this direction, we have extended the normative language recently proposed in [20]. That proposal is enriched with new types of norms, namely norms that we keep active during a time interval, and conditional norms over the institutional state, (e.g. the observable attributes of agents and objects of the environment). Moreover, our extension of that language includes the possibility to sanction agents by modifying their institutional state, i.e. their observable attributes. Nonetheless, since in EIs alls actions are speech acts, actions expressed by the language are limited to the utterance of illocutions.

We propose the BNF description of our normative language as follows:

$$\begin{aligned}
NORM &:= N(\ utter(S,W,I)\ \langle TIME\rangle\ \langle \textsf{IF}\ C\rangle\ ) \\
N &:= \textsf{OBLIGED} \mid \textsf{PERMITTED} \mid \\
&\quad\ \textsf{FORBIDDEN} \\
I &:= \iota(A,R,A,R,M,T) \\
TIME &:= \textsf{BEFORE}\ D \mid \textsf{AFTER}\ D \mid \\
&\quad\ \textsf{BETWEEN}\ (D,D) \mid \\
&\quad\ \textsf{BEFORE}\ uttered(S^*,W^*,I^*) \mid \\
&\quad\ \textsf{AFTER}\ uttered(S^*,W^*,I^*) \mid \\
&\quad\ \textsf{BETWEEN}\ (\ uttered(S^*,W^*,I^*), \\
&\qquad\qquad\qquad uttered(S^*,W^*,I^*)\ ) \\
C &:= \neg\ (CONDS) \mid CONDS \\
CONDS &:= \langle\neg\rangle COND\ \langle, C\rangle \\
COND &:= V\ OP\ V \mid uttered(S^*,W^*,I^*) \mid \\
&\quad\ N(\ utter(S^*,W^*,I^*)\ ) \mid predicate \\
V &:= AT \mid F \mid value \\
AT &:= identifier.attribute \mid variable \\
OP &:= \ > \mid < \mid \geq \mid \leq \mid = \\
SANCTION &:= \textsf{SANCTION}\ (\ (COMMS)\ \textsf{IF}\ NP\ (NORM)\ ) \\
NP &:= \textsf{VIOLATED} \mid \textsf{COMPLIED} \\
COMMS &:= COMM\langle, COMMS\rangle \\
COMM &:= AT\ =\ F \mid F \\
F &:= identifier(\ < ARGS >\ ) \\
ARGS &:= V\ \ <,\ V >
\end{aligned}$$

where S is a scene identifier; W is a state identifier; ι is an illocutionary particle; A is an agent identifier; R is a role identifier; M is a content message in the language L_O from the dialogical framework; T is a time stamp; D is a deadline; $S\ , W\ , I\ , A\ , R\ , M\ , T$ are expressions which may contain variables referring, respectively, to scenes, states, illocutions, agent identifiers, role identifiers, messages and time stamp; and *predicate* is a first-order formula whose variables are universally quantified.

On the one hand, $utter(s\ ,w\ ,i\ )$ is the predicate that represents the action (not carried out yet) of submitting an illocution at the state w of scene s . This predicate is the only one that can be restricted with deontic operators. On the other hand, $uttered(s\ ,w\ ,i\ )$ is used to denote that the submission of an illocution has been carried out. The latter predicate can be used in the conditional construct of a normative rule.

From the BNF notation follows that a norm ($NORM$) can be either an obligation (OBLIGED), a permission (PERMITTED) or a prohibition (FORBIDDEN) upon the utterance of a given illocution ($utter(S, W, I)$) if conditions are satisfied (IF C). The BEFORE construct is used to activate the norm before a deadline or an action. The AFTER construct is used to activate the norm after a given deadline or an action. The BETWEEN construct results from the combination of the previous two and it is used to activate the norm once the time specified by the first argument is reached and de-activate it once the time specified by the second argument is reached. The IF construct is used to introduce conditions over variables, agents' observable attributes or function results. The AT definition notates how attribute values can be accessed with the language, *identifier.attribute* denotes that the value of the attribute with name *attribute* of the agent or object with name *identifier* is retrieved.

Sanctions (analogously, rewards) can also be expressed by defining the sequence of attribute updates or functions ($COMMS$) to be executed if a norm is violated (analogously, complied) (VIOLATED $NORM$ or COMPLIED $NORM$).

3.1 Examples

In this section we show how to use our normative language through several examples. All these examples, along with the rest of examples in this paper, are based on an electronic auction institution. The institution has four scenes or activities: *Registration*, where agents sign in along with the information about the goods they want either buy or sell; *Auction*, where the actual bidding takes place; *Payment*, where buyers pay for acquired items and sellers are paid; and *Delivery*, where the sold goods are delivered to the acquiring buyers.

```
OBLIGED  (utter(payment, W,
                inform(A, buyer, B, payee, pay(IT, P)))
 BEFORE  uttered(payment, w5,
                inform(B, payee, all, buyer,
                     close()))
     IF  uttered(auction, w2,
                inform(A, auctioneer, all, buyer,
                     sold(IT, P, C))),
         A.credit > P
```

Figure 1: Conditional obligation with deadline

After the registration of agents and goods, agents join the *Auction* scene to start a Dutch auction. Initially, the auctioneer agent informs all buyers about the good being auctioned along with its initial price. The auctioneer progressively decreases the call price until a bidder stops the clock. If the good has not been sold when the call price reaches the *reserve price*, the auction finishes off and the good is withdrawn. If there is a bid collision, i.e. more than one bid is submitted at the same time, the call price is increased and a new round is started. If only an agent places the bid during a round and has enough credit, it wins the auction. When an agent wins an auction it must proceed to the *Payment* scene to pay for the purchased goods. After the payment of goods, an agent taking on the *storemanager* role must deliver them to the buyer before a deadline.

```
PERMITTED  (utter(auction, W,
                  inform(A, buyer, B, auctioneer, bid(IT, P)))
  BETWEEN  (uttered(auction, w0,
                  inform(B, auctioneer, all, buyer,
                       offer(IT, P)))
           uttered(auction, w2,
                  inform(B, auctioneer, all, buyer,
                       sold(IT, P, C))))
```

Figure 2: Permission in an interval of time

Figure 1 contains a conditional obligation with deadline. Intuitively, it means that if a buyer submitted a winning bid for a good, he must pay for it before the payment scene is closed: If agent A, playing the *buyer* role, submits to agent C, playing the *auctioneer* role, a bid for a good at price P and agent A's credit is greater than P, then A is obliged to pay in the *payment* scene to an agent playing the *payee* role before the latter closes the scene.

Figure 2 shows a permission that is active during a time interval. Its intuitive meaning is that a buyer is permitted to bid after hearing an offer but before the auctioneer declares the sale: Agent A playing the *buyer* role is permitted to submit a bid for an item IT to agent B playing the *auctioneer* role in *auction* scene after B informs all buyers of an offer but before B informs all buyers of the sale.

Figure 3 shows a sanction on an agent's credit when a deadline can not be met. If agent A is obliged to inform about the delivery of an item before a deadline and the agent does not meet the deadline, his credit is reduced by ten units.

```
   SANCTION  (A.credit = A.credit − 10
IF VIOLATED  ( OBLIGED(utter(S, W,
                         inform(A, R1, B, R2, deliver(IT)))
               BEFORE 15/10/05) ) )
```

Figure 3: Sanction related to a deadline violation

4. EXECUTABLE NORMS

Once the normative language has been defined, we need to handle the normative state of an institution. A rule-based system was chosen to implement norms because the normative language is of the form *preconditions* $\leadsto$ *postconditions*, which is easily expressable with rules. In order to facilitate the integration with AMELI we decided to implement this tool with Jess since both are written in Java.

In this section we first introduce the (norm) engine used for implementing executable norms. The translation of norms expressed in the language presented in section 3 into executable norms written in Jess will be also detailed.

4.1 Jess

Jess is an expert system shell and scripting language from Sandia National Laboratories [13] written entirely in Java [10]. Jess supports the development of rule-based systems that can be tightly coupled to code written in Java. It can manipulate Java objects and can be extended with new functions implemented in Java.

4.1.1 Facts

A rule-based system maintains a collection of knowledge portions called facts. This collection is known as the knowledge base. In Jess, there are three kinds of facts: ordered facts, unordered facts, and definstance facts. Ordered facts are simply Lisp-style lists where the first field, the head of the list, acts as a category for the fact. Unordered facts allow the programmer to structure the properties of a fact in slots. Before the creation of unordered facts, the slots they have must be defined using the *deftemplate* construct.

Figure 4 shows an example of an unordered fact template used to model the predicate *uttered*. An `uttered` fact is composed of several slots: `scene`, `state`, `agent`, `receiver`, `performative` and `content`. The scene and state where an utterance takes place is specified by the `scene` and `state` slot; while the `agent` and `receiver` slots define the sender and receiver of the message (`content`). The illocutionary particle of the illocution is stated by the `performative` slot.

```
(deftemplate uttered
    (slot scene)
    (slot state)
    (slot agent)
    (slot receiver)
    (slot performative)
    (multislot content))
```

Figure 4: Example of a Jess unordered fact

4.1.2 Rules

Rules have two parts separated by the connective =¿ : a left-hand side (LHS) and a right-hand side (RHS). The LHS is employed for matching fact patterns. The RHS is a list of actions (postconditions) to perform if the patterns of the LHS (preconditions) are satisfied. These actions are typically method calls. An important feature of Jess is that the RHS can call native Jess methods, instance methods of externally referenced Java objects and static class methods. This feature adds enormous flexibility to the code.

```
(defrule cob-1-sanction
    "Reduce agent's credit on violation"
     (V  (type negative) (constraints ?c) (agent ?a)
         (scene deliver) (state w0) (receiver ?b)
         (performative inform) (content deliver ?it))
     (agent (id ?a) (attrs ?at ))
     =>
        (bind ?old (?at get "credit"))
        (bind ?new (- ?old 100))
        (?at put "credit" ?new))
```

Figure 5: Example of a Jess rule

Figure 5 shows an example of a rule. When a violation occurs, that is, when exists a fact `V` with the specified slots and the attributes of the violator agent `?a` can be retrieved in the variable `?at` then store the value of the credit of the agent in variable `?old`, store in `?new` the value of variable `?old` decreased by a hundred and change the credit of the violator agent `?a` into the value of variable `?new`.

4.2 Norm implementation

In addition to the normative language we need to keep at run-time the sequence of done actions and to query what actions are permitted or forbidden and what are the pending obligations. To introduce utterances, permissions, prohibitions and obligations in the norm engine, a translation from our language to Jess rules is needed.

This translation can be carried out using the criteria established in the following sections.

We define four types of Jess unordered facts: `O`, `P`, `F` and `V` that stand, respectively, for obligations, permissions, prohibitions and violations.

4.2.1 Conditional norms.

Conditional norms are those norms that include an IF section. The translation of IF sections is directly realised by placing the conditions in the LHS of a Jess rule.

OBLIGED(	$utter(delivery, w0,$
	$inform(C, storemanager, A, payer, deliver(IT)))$
BEFORE	$15\ days$
IF	$uttered(payment, w0,$
	$inform(A, payer, B, payee, pay(IT, P))))$

Figure 6: Example of a conditional obligation with deadline

```
(defrule cob-1
   (uttered (agent ?a) (scene payment)
            (state w0) (receiver payee)
            (performative inform)
    (content pay ?it ?price ))
  =>
    (assert (O (agent storemanager) (scene delivery)
               (state w0) (receiver ?a)
               (performative inform)
       (content deliver ?it)))
    (bind ?date (new java.util.Date))
    (bind ?deadline (add-date ?date 0 0 15 0 0 0 0))
    (bind ?rule (str-cat
         "(defrule cob-1-deadline "
              "(not(uttered (agent payee)"
              "(scene delivery)"
              "(state w0) (receiver " ?a ")"
              "(performative inform)"
              "(content deliver " ?it ") ) )"
         " =>  "
         "(assert (V (type negative) "
              "(constraints \"before "
                  (?deadline toString) "\")"
              "(agent payee) (scene deliver)"
              "(state w0) (receiver " ?a ")"
              "(performative inform)"
              "(content deliver " ?it "))))"))

    (set-deadline ?deadline ?rule ) )
```

Figure 7: Implementation in Jess of a conditional obligation with deadline

4.2.2 Action-dependent norms.

Action-dependent norms are those norms that include a BEFORE, AFTER or BETWEEN section followed by an ac-

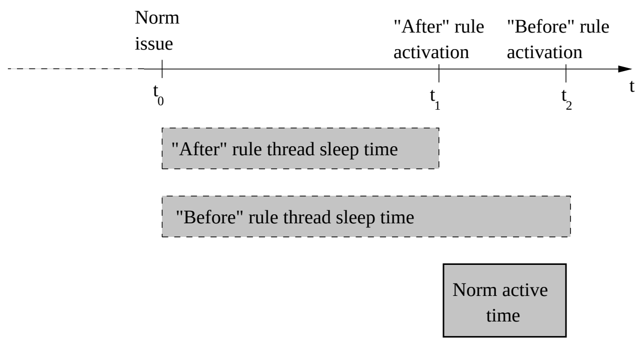

Figure 8: Time diagram of rule activation for norm OBLIGED($utter(s, w, i)$ BETWEEN t_1, t_2)

tion (as shown in figure 2). To translate an obligation to be fulfilled before the utterance of an illocution i_1, we add a rule that asserts a violation fact if illocution i_1 has been uttered but the obliged illocution has not. The assertion of facts can be achieved with the Jess function `(assert ?fact)`. The translation of permissions or prohibitions that are active before the utterance of an illocution i_1 occurs is made by asserting the given permission or prohibition and adding to the Jess engine a rule that retracts it when illocution i_1 is uttered.

In order to translate obligations, permissions and prohibitions that are active after the utterance of a given illocution i_2; we add a rule that asserts the obligation, permission or prohibition when i_2 is uttered.

The translation of permissions, prohibitions and obligations during a time interval (BETWEEN construct) is a combination of the three previous cases. We decompose the BETWEEN construct into two Jess rules as if it had an AFTER and BEFORE constructs. The translation of these constructs is carried out as stated above.

4.2.3 Time-dependent norms.

Time-dependent norms are those norms that include a BEFORE, AFTER or BETWEEN section followed by a date.

To translate rules with temporal constraints (i.e. the BEFORE,
AFTER and BETWEEN constructs with time objects) into Jess rules we use the user-defined function `(set-deadline ?deadline ?rule)` where *?deadline* is an absolute date object indicating when the rule fires and *?rule* is a string-based representation of a rule. In this way the `set-deadline` function adds the given rule to the Jess engine only when the specified absolute date arrives.

To translate obligations with deadline (BEFORE construct), we use the `set-deadline` function to add a Jess rule that asserts a violation when the deadline has not been met. In other words, it checks, after the deadline, if the obliged illocution has not been uttered yet, in order to fire the corresponding violation.

The translation of permissions and prohibitions that are active before a deadline is done by asserting the permission or prohibition and setting a deadline rule that retracts the permission or prohibition when the deadline has passed.

Figures 6 and 7 show an example of the translation of a conditional obligation with a deadline into a Jess rule. Their intuitive meaning is that paid goods must be delivered before 15 days. If agent A playing the *payer* role pays for an item to an agent B playing the *payee* role in the *payment* scene, an agent playing the *storemanager* role must deliver that item to the purchaser in the *delivery* scene before 15 days.

To translate obligations, permission or prohibitions that activate after a deadline, we add a deadline rule that asserts the obligation, permission or prohibition after the deadline.

OBLIGED(	$utter(deliver, w0,$
	$inform(C, storemanager, A, buyer, deliver(IT)))$
BETWEEN	$3\ day,\ 15days$
IF	$uttered(payment, w0,$
	$inform(A, payer, B, payee, pay(IT, P))))$

Figure 9: Conditional obligation along a time interval

For this purpose we use the `set-deadline` function to add a Jess rule that asserts the obligation, permission or prohibition once the deadline has passed.

Finally, obligations, permissions and prohibitions during a time interval can be translated as a combination of the previous two cases: we add a rule for the AFTER construct and another one for the BEFORE construct.

Figure 8 depicts the time diagram of a rule with a BETWEEN construct which is translated into two Jess rules that activate at times t_1 and t_2.

```
(defrule obt-1
    (uttered (agent ?a) (scene payment)
             (state w0) (receiver payee)
             (performative inform)
             (content pay ?it ?price ))
  =>
    (bind ?date (new java.util.Date))
    (bind ?t1 (add-date ?date 0 0 3 0 0 0 0))
    (bind ?t2 (add-date ?date 0 0 15 0 0 0 0))

    (bind ?rule1 (str-cat
      "(defrule obt-1-after => "
      "(assert (O (agent storemanager) (scene deliver) "
      "(state w0) (receiver " ?a ")"
      "(performative inform)
      "(content deliver it))))" ))

    (bind ?rule2 (str-cat
      "(defrule obt-1-before =>"
      "(assert (V (type negative)"
      "(constraints \"before "
            (?t2 toString) "\")"
      "(agent storemanager) (scene deliver)"
      "(state w0) (receiver " ?a ")"
      "(performative inform)"
      "(content deliver it) )))" ))

    (set-deadline ?t1 ?rule1 )
    (set-deadline ?t2 ?rule2 ))
```

Figure 10: Implementation in Jess of a conditional obligation along a time interval

Figures 9 and 10 show a compound norm that has conditional and temporal sections. In figure 9 the action dependence of the norm is expressed in the conditional section. In figure 10 the time dependence is described by the BETWEEN construct. They oblige a store manager agent to deliver the goods between 3 to 15 days after the sale date. Figure 10 shows the translation of the normative rule in 9 into a Jess rule.

5. CONCLUSIONS AND FUTURE WORK

We have defined a normative language to specify obligations, permissions, prohibitions, violations and sanctions to restrict agents' dialogical actions. This normative language can be used as an extension of the normative rules of the current version of electronic institutions obtaining a higher degree of expressiveness and flexibility. We have also implemented a norm engine which mantains the normative state of an institution, i.e. the permissions, prohibitions and pending obligations that hold in the current state of execution.

There are some differences between our normative proposal and other recent ones, the more salient are that we do not need norms over predicates since we have assumed that all admissible actions within an electronic institutions are speech acts (as in for example, [18][19][6]), we do not make the strong assumption (as in, for example, [20]) that there is a prohibition before an action iff there is a permission after that action, and we do have a working proof of concept implementation of a computationally feasible framework. With respect to other implementation proposals for normative frameworks, ours is more expressive in the sense that other implementations do not include temporal aspects in the definition of norms and, in the test of conditional norms or in the application of sanctions, fail to consider observable agent attributes or attributes of objects in the environment.

As far as future work is concerned, we intend to produce an upward compatible extension of the EIDE environment through the addition of an automatic translation module to map our normative language into Jess rules and integrates our norm engine with AMELI. We are extending the notions stated above in the formalization and development of a norm-based agent programming language [9]. We leave as future work the analysis of the expressiveness of our language in comparison to another recent approaches as [8] and [4].

Acknowledgments

The present paper was funded by the Spanish Science and Technology Ministry as part of the Web-i-2 project (TIC-2003-08763-C02-00). A. Garcia-Camino enjoys an I3P grant of the Spanish Council for Scientific Research (CSIC).

6. REFERENCES

[1] A. Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, November 2003.

[2] A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A protocol for resource sharing in norm-governed ad hoc networks. In *Proceedings of the Declarative Agent Languages and Technologies (DALT) workshop*, July 2004.

[3] J. Broersen, F. Dignum, V. Dignum, and J.-J. C. Meyer. Designing a deontic logic of deadlines. In *7th Int. Workshop of Deontic Logic in Computer Science (DEON'04)*, pages 43–56, Portugal, May 2004.

[4] H. L. Cardoso and E. Oliveira. Virtual enterprise normative framework within electronic institutions. In *Proceedings of the Fifth International Workshop Engineering Societes in the Agents World (ESAW)*, 2004.

[5] F. Dignum, J. Broersen, V. Dignum, and J.-J. C. Meyer. Meeting the deadline: Why, when and how. In *3rd Int. Workshop on Formal Approaches to Agent-Based Systems (FAABS)*, pages 30–40, Maryland, April 2004.

[6] M. Esteva. *Electronic Institutions: from specification to development.* Number 19 in IIIA Monograph Series. PhD Thesis, 2003.

[7] M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 236–243, 2004.

[8] N. Fornara, F. Viganò, and M. Colombetti. A communicative act library in the context of artificial institutions. In *Second European Workshop on Multi-Agent Systems*, pages 223–234, Barcelona, 2004.

[9] A. Garcia-Camino, J.A.Rodriguez-Aguilar, C. Sierra, and W. Vasconcelos. A distributed architecture for norm-aware agent societies. In *Proceedings of the Declarative Agent Languages and Technologies (DALT) workshop*, Utrecht, July 2005.

[10] J. Gossling. *The Java programming Language.* Reading. Addison-Wesley, 1996.

[11] C. Hewitt. Offices are open systems. *ACM Transactions of Office Automation Systems*, 4(3):271–287, 1986.

[12] N. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems*, 1:7–38, 1998.

[13] Jess. Jess,the rule engine for java. Sandia National Laboratories. http://herzberg.ca.sandia.gov/jess, November 2004.

[14] A. Lomuscio and D. Nute, editors. *Proc. of the 7th Int. Workshop on Deontic Logic in Computer Science (DEON'04)*, volume 3065. Springer Verlag, 2004.

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

[16] F. López y López, M. Luck, and M. d'Inverno. Constraining autonomy through norms, 2002.

[17] L. Michael, D. C. Parkes, and A. Pfeffer. Specifying and monitoring market mechanisms using rights and obligations. In *Proc. AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC VI)*, New York, USA, 2004.

[18] P. Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor.* Number 8 in IIIA Monograph Series. PhD Thesis, 1997.

[19] J. A. Rodriguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions.* Number 14 in IIIA Monograph Series. PhD Thesis, 2001.

[20] J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Norms in multiagent systems: some implementation guidelines. In *Second European Workshop on Multi-Agent Systems*, pages 737–748, Barcelona, 2004.