Ignoring, Forcing and Expecting Concurrent Events in Electronic Institutions

Andrés García-Camino

IIIA, Artificial Intelligence Research Institute
CSIC, Spanish National Research Council
Campus UAB, 08193 Bellaterra, Spain
andres@iiia.csic.es

Abstract. Norms constitute a powerful coordination mechanism among heterogeneous agents. We propose means to specify open environments regulated using the notions of ignoring, forcing, expecting and sanctioning events and prevention of unwanted states. These notions make explicit and clear the stance of institutions about forbidden and obligatory behaviour. Our rule-based language calculates the effects of concurrent events generated by agents given a set of norms based on the deontic notions previously mentioned. Our formalism has been conceived as basis for an implementation of Electronic Institutions.

1 Introduction

Ideally, open multi-agent systems (MAS) involve heterogeneous and autonomous agents whose concurrent 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 are many examples of languages for regulating agent behaviour (for example, [1–5]). However, very few of them regulate concurrent events taking into account the rest of events that occur at an instant of time. The few that exist (e.g. [3]) are not conceived to deal with open MAS.

Furthermore, in the literature we find that almost all these languages are based on deontic logic [6] that establishes which actions are permitted, forbidden or obligatory. However, it does not establish which is the semantics of these modalities with respect to a computational system. For instance, when an action is claimed to be forbidden, does it means that it is prevented to happen, or that the agents that bring it about must be sanctioned or that the effects of that action are just ignored?

Instead, we propose a language, called $\mathcal{I}$, and one implementation of it that uses the notions of ignoring, forcing, and expecting events along with the notion of preventing a state, in the computation of the effects of concurrent agent behaviour in a regulated open MAS. The main contributions of $\mathcal{I}$ is the management of sets of events that occur simultaneously and the distinction between

norms that can be violated or not. For instance, an obligation that may be violated to perform a set of simultaneous events is represented as the expectation of the attempts to perform them. However, the enforcement of an obligation that may not be violated to perform a set of events is carried out by the system by taking these events as performed even they are not. We denote such enforcement as forcing events.

The paper is structured as follows. Section 2 introduces $\mathcal{I}$, a rule language for electronic institutions. A basic example illustrating the expressiveness of $\mathcal{I}$ is shown in section 3. In section 4, we introduce the formulae that we use for modelling electronic institutions. An example of a bank institution is presented in section 5. In section 6 we contrast our approach with a sample of other contemporary work. Finally, we draw conclusions and outline future work in section 7.

2 $\mathcal{I}$: A Rule Language for Electronic Institutions

In this section we introduce a rule language for the regulation and management of concurrent events generated by a population of agents. Our rule-based language allows us to represent norms and changes in an elegant way.

The building blocks of our language are first-order terms and implicitly, universally quantified atomic formulae without free variables. We shall make use of numbers and arithmetic functions to build terms; arithmetic functions may appear infix, following their usual conventions[1]. 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.

$$
\begin{array}{rl}
\textit{ECA-Rule} & ::= \textbf{on } \textit{events } \textbf{if } \textit{conditions } \textbf{do } \textit{actions} \\
\textit{if-Rule} & ::= \textbf{if } \textit{conditions } \textbf{do } \textit{actions} \\
\textit{ignore-Rule} & ::= \textbf{ignore } \textit{events } \textbf{if } \textit{conditions} \\
\textit{prevent-Rule} & ::= \textbf{prevent } \textit{conditions } \textbf{if } \textit{conditions} \\
\textit{force-Rule} & ::= \textbf{force } \textit{events } \textbf{on } \textit{events } \textbf{if } \textit{conditions } \textbf{do } \textit{actions} \\
\textit{events} & ::= \textit{list\_of\_events} \mid \emptyset \\
\textit{list\_of\_events} & ::= \textit{atomic\_formula}, \textit{list\_of\_events} \mid \textit{atomic\_formula} \\
\textit{conditions} & ::= \textit{conditions} \wedge \textit{conditions} \mid \neg(\textit{conditions}) \mid \textit{atomic\_formula} \\
\textit{actions} & ::= \textit{action} \bullet \textit{actions} \mid \textit{action} \\
\textit{action} & ::= \oplus \textit{atomic\_formula} \mid \ominus \textit{atomic\_formula}
\end{array}
$$

Fig. 1. Grammar for $\mathcal{I}$

One goal of the $\mathcal{I}$ language is to specify which are the effects of concurrent events and this is achieved with Event-Condition-Action (ECA) rules. Intuitively, an ECA-rule means that whenever the events occur and the conditions hold then the actions are applied. These actions consist in the addition and removal of atomic formulae from the state of affairs. ECA-rules are checked in parallel and they are executed only once without chaining.

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

If-rules are similar to rules in standard production systems, if the conditions hold then the actions are applied. They are implemented with a forward chaining mechanism: they are executed sequentially until no new formula is added or removed.

Ignore-rules are used for ignoring events when the conditions hold in order to avoid unwanted behaviour. Similarly, prevent-rules are used for preventing some conditions to hold in the situations given. In order to prevent unwanted states, events causing such unwanted states are ignored. Force-rules generate events and execute actions as consequence of other events and conditions.

Sanctions over unwanted events can be carried out with ECA-rules. For instance, we can decrease the credit of one agent by 10 if she generates certain event.

We add an additional kind of rules, expectation-rules, that generate and remove expectations of events. If the expectation fails to be fulfilled then some sanctioning or corrective actions are performed.

$$\begin{aligned} \textit{expectation-Rule} ::= \ & \textbf{expected } \textit{event} \textbf{ on } \textit{events} \textbf{ if } \textit{conditions} \\ & \textbf{fulfilled-if } \textit{conditions}' \textbf{ violated-if } \textit{conditions}'' \\ & \textbf{sanction-do } \textit{actions} \end{aligned}$$

Each expectation rule can be translated into three ECA-rules:

$$\textbf{on } \textit{events} \textbf{ if } \textit{conditions} \textbf{ do } \oplus \textit{exp}(\textit{event}) \tag{1}$$

$$\textbf{if } \textit{exp}(\textit{event}) \wedge \textit{conditions}' \textbf{ do } \ominus \textit{exp}(\textit{event}) \tag{2}$$

$$\textbf{if } \textit{exp}(\textit{event}) \wedge \textit{conditions}'' \textbf{ do } \ominus \textit{exp}(\textit{event}) \bullet \textit{actions} \tag{3}$$

Rules 1 and 2 respectively adds and removes an expectation whenever the events have occurred and the conditions hold. Rule 3 cancels the unfulfilled expectation and sanctions an agent for the unfulfilled expectation by executing the given *actions* whenever some *conditions* hold.

2.1 Semantics

Instead of basing the $\mathcal{I}$ language in the standard deontic notions, two types of prohibitions and two types of obligations are included. In our language, ECA-rules determine what is possible to perform, i.e. they establish the effects (including sanctions) in the institution after performing certain (possibly concurrent) events. ECA-rules might be seen as conditional count-as rules: the given events count as the execution of the actions in the ECA-rule if the conditions hold and the event is not explicitly prohibited. As for the notion of permission, all the events are permitted if not explicitly prohibited. The notion of an event being prohibited may be expressed depending on whether that event has to be ignored or not. If not otherwise expressed, events are not ignored. Likewise, the notion of a state being prohibited may be specified depending on whether that state has to be prevented or not. By default, states are not prevented. Obligations are differentiated in two types: expectations, which an agent may not fulfill, and forced (or obligatory) events, which the system takes as institutional events even they are not really performed by the agents.

Each set of ECA-rules generates a labelled transition system $\langle S, E, R\rangle$ where each state S is a set of atomic formulae, E is a set of events, and R is a $S \times 2^E \times S$ relationship indicating that whenever a set of events occur in the former state, then there is a transition to the subsequent state.

Ignore-rules avoid to execute any transition that contains in its labelling all the events that appear in each ignore-rule. For instance, having a rule **ignore** α_1 **if** *true* would avoid to execute the transitions labelled as $\{\alpha_1\}$, $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_2, \alpha_3\}$. However, having a rule **ignore** α_1, α_2 **if** *true* would avoid to execute $\{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_2, \alpha_3\}$ but not $\{\alpha_1\}$.

Prevent-rules ignore all the actions in an ECA-rule if it brings the given formulae about. For example, suppose that we have

$$\textbf{prevent } q_1 \textbf{ if } \textit{true}$$

along with ECA-rules 4, 5 and 6. After the occurrence of events α_1 and α_2 and since q_1 is an effect of event α_2, all the actions in ECA-rule 5 would be ignored obtaining a new state where p and r hold but neither q_1 nor q_2.

$$\textbf{on } \alpha_1 \textbf{ if } \textit{true} \textbf{ do } \oplus p \tag{4}$$

$$\textbf{on } \alpha_2 \textbf{ if } \textit{true} \textbf{ do } \oplus q_1 \bullet \oplus q_2 \tag{5}$$

$$\textbf{on } \alpha_1, \alpha_2 \textbf{ if } \textit{true} \textbf{ do } \oplus r \tag{6}$$

Force-rules generate events during the execution of the transition system. However, the effects of such events are still specified by ECA-rules and subject to prevent and ignore-rules.

2.2 Operational Semantics

In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation [7, 8]:

We now define the semantics of the conditions, that is, when a condition holds:

Definition 1. *Relation* $\mathbf{s}_l(\Delta, C, \sigma)$ *holds between state* Δ*, a condition* C *in an* **if** *clause and a substitution* σ *depending on the format of the condition:*

1. $\mathbf{s}_l(\Delta, C \wedge C', \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, C, \sigma')$ *and* $\mathbf{s}_l(\Delta, C' \cdot \sigma', \sigma'')$ *hold and* $\sigma = \sigma' \cup \sigma''$.
2. $\mathbf{s}_l(\Delta, \neg C, \sigma)$ *holds iff* $\mathbf{s}_l(\Delta, C, \sigma)$ *does not hold.*
3. $\mathbf{s}_l(\Delta, seteq(L, L2), \sigma)$ *holds iff* $L \subseteq L2$*,* $L2 \subseteq L$ *and* $\mid L \mid = \mid L2 \mid$.
4. $\mathbf{s}_l(\Delta, true, \sigma)$ *always holds.*
5. $\mathbf{s}_l(\Delta, \alpha, \sigma)$ *holds iff* $\alpha \cdot \sigma \in \Delta$.

Case 1 depicts the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction. Case 2 introduces the negation by failure. Case 3 compares if two lists have the same elements possibly in different order. Case 4 gives semantics to the keyword "*true*". Case 5 holds when an atomic formulae α is part of the state of affairs.

We now define the semantics of the actions of a rules:

Definition 2. *Relation* $\mathbf{s}_r(\Delta, A, \Delta')$ *mapping a state* Δ*, the action section of a rule and a new state* Δ' *is defined as:*

1. $\mathbf{s}_r(\Delta, (A \bullet As), \Delta')$ *holds iff both* $\mathbf{s}_r(\Delta, A, \Delta_1)$ *and* $\mathbf{s}_r(\Delta_1, As, \Delta')$ *hold.*
2. $\mathbf{s}_r(\Delta, \oplus\alpha, \Delta')$ *holds iff*
   (a) $\alpha \notin \Delta$ *and* $\Delta' = \Delta \cup \{\alpha\}$ *or;*
   (b) $\Delta' = \Delta$.
3. $\mathbf{s}_r(\Delta, \ominus\alpha, \Delta')$ *holds iff*
   (a) $\alpha \in \Delta$ *and* $\Delta' = \Delta \setminus \{\alpha\}$ *or;*
   (b) $\Delta' = \Delta$.

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 and 3 cater respectively for the insertion and removal of atomic formulae α.

We now define relation $check_{prv}$ that checks if there is no prevent-rule that has been violated, i.e., not all the conditions hold in the state of affairs Δ. It checks whether Δ contain all the conditions of each prevent-rule or not.

Definition 3. *Relation* $check_{prv}(\Delta, PrvRules)$ *mapping a state* Δ *and a sequence PrvRules of prevent-rules holds iff an empty set is the largest set of conditions* C *such that prevent-rule* $p =$ **prevent** C **if** C', $p \in PrvRules$, $\mathbf{s}_l(\Delta, C)$ *and* $\mathbf{s}_l(\Delta, C')$ *hold.*

Definition 4. *Relation* $fire(\Delta, PrvRules, \textbf{if } C \textbf{ do } A, \Delta')$ *mapping a state* Δ*, a sequence PrvRules of prevent-rules, an if-rule and a new state* Δ' *holds iff* $assert(fired(C))$, $\mathbf{s}_r(\Delta, A, \Delta')$ *and* $check_{prv}(\Delta', PrvRules)$ *hold.*

Relation *can_fire* checks whether the conditions of a given if-rule hold and the rule after applying substitution σ has not been already fired.

Definition 5. *Relation* $can\_fire(\Delta, \textbf{if } C \textbf{ do } A, \sigma)$ *mapping a state* Δ *an if-rule and a substitution* σ *holds iff* $\mathbf{s}_l(\Delta, C, \sigma)$ *holds and* $fired(C \cdot \sigma)$ *does not hold.*

Relation *resolve* determines the rule that will be fired by selecting the first rule in the list.

Definition 6. *Relation* $resolve(RuleList, SelectedRule)$ *mapping a list of if-rules and a selected if-rule holds iff*

1. $RuleList = \emptyset$ *and* $SelectedRule = \emptyset$*; or*
2. $RuleList = \langle r_1, \cdots, r_n \rangle$ *and* $SelectedRule = r_1$.

Relation *select_rule* determines the rule that will be fired by selecting all the rules that can fire and resolving the conflict with relation *resolve*.

Definition 7. *Relation* $select\_rule(\Delta, IfRulesList, SelectedRule)$ *mapping a state of affairs* Δ *a list of if-rules and a selected if-rule holds iff Rs is the largest set of rules* $R \in IfRulesList$ *such that* $can\_fire(\Delta, R, \sigma)$*;* $resolve(Rs, SR)$ *hold and* $SelectedRule = SR \cdot \sigma$.

Relation $\mathbf{s}_{if}$ determines the new state of affairs after applying a set of if-rules to a initial state of affairs taking into account a set of prevent-rules.

Definition 8. *Relation* $\mathbf{s}_{if}(\Delta, IfRules, PrvRules, \Delta')$ *mapping a state of affairs* Δ*, a list of if-rules, a list of prevent-rules and a new state of affairs holds iff*

1. *select_rule*$(\Delta, IfRules, R)$ *hold,* $R \neq \emptyset$*,* *fire*$(\Delta, PrvRules, R, \Delta'')$ *and* $\mathbf{s}_{if}(\Delta'', IfRules, PrvRules, \Delta')$ *hold; or*
2. *select_rule*$(\Delta, IfRules, R)$ *hold,* $R = \emptyset$*; or*
3. $\mathbf{s}_{if}(\Delta, IfRules, PrvRules, \Delta')$ *hold.*

Relation *ignored* determines a set of events that occurred have to be ignored taking into account a list of ignore-rules.

Definition 9. *Relation* $ignored(\Delta, \Xi, E, IgnRules)$ *mapping a state of affairs* Δ*, a list* Ξ *of events that occurred, a list of events in a ECA-rule and a list of ignore-rules holds iff* $i =$ **ignore** E' **if** C*,* $i \in IgnRules$*,* $E' \subseteq \Xi$*,* E *intersects with* E' *and* $\mathbf{s}_l(\Delta, C)$ *holds.*

Relation $\mathbf{s}'_r$ applies $\mathbf{s}_r$ first and then $\mathbf{s}_{if}$ in order to activate the forward chaining.

Definition 10. *Relation* $\mathbf{s}'_r(\Delta, IfRules, PrvRules, ActionList, \Delta')$ *mapping a state of affairs* Δ*, a list of if-rules, a list of prevent-rules, a list of actions and a new state of affairs holds iff*

1. $ActionList = \emptyset$ *and* $\Delta' = \Delta$*; or*
2. $ActionList = \langle a_1, \cdots, a_n \rangle$*,* $\mathbf{s}_r(\Delta, a_1, \Delta'')$*,* $check_{prv}(\Delta'', PrvRules)$*,* $\mathbf{s}_{if}(\Delta'', IfRules, PrvRules, \Delta''')$ *and* $\mathbf{s}'_r(\Delta''', IfRules, PrvRules, \langle a_2, \cdots, a_n \rangle, \Delta')$ *hold; or*
3. $\mathbf{s}'_r(\Delta, IfRules, PrvRules, \langle a_2, \cdots, a_n \rangle, \Delta')$*.*

Relation $\mathbf{s}_{on}$ calculates the new state of affairs Δ' from an initial state Δ and a set Ξ of events that occurred applying a list of ECA-rules, if-rules, ignore-rules and prevent-rules.

Definition 11. *Relation* $\mathbf{s}_{on}(\Delta, \Xi, ECARules, IfRules, IgnRules, PrvRules, \Delta')$ *mapping a state of affairs* Δ*, a list* Ξ *of events that occurred, a list of ECA-rules, a list of if-rules, a list of ignore-rules, a list of prevent-rules, and a new state of affairs holds iff* As *is the largest set of actions* $A' = A \cdot \sigma$ *in a ECA-rule* $r =$ **on** E **if** C **do** A *such that* $R \in ECARules$*,* $E \cdot \sigma' \subseteq \Xi$*,* $\mathbf{s}_l(\Delta, C, \sigma'')$ *hold,* $ignored(\Delta, \Xi, E, IgnRules)$ *does not hold and* $\sigma = \sigma' \cup \sigma''$*; and* $\mathbf{s}'_r(\Delta, IfRules, PrvRules, As, \Delta')$ *hold.*

Relation $\mathbf{s}_f$ calculates the new state of affairs Δ' and the new set Ξ' of occurred events from an initial state Δ and a set Ξ of events that occurred applying a list of if-rules, ignore-rules, prevent-rules and force-rules.

Definition 12. *Relation* $\mathbf{s}_f(\Delta, \Xi, IfRules, IgnRules, PrvRules, FrcRules, \Xi', \Delta')$ *mapping a state of affairs* Δ*, a list* Ξ *of events that occurred, a list of if-rules, a list of ignore-rules, a list of prevent-rules, a list of force-rules, a new list of events that occured and a new state of affairs holds iff EAs is the largest set of tuples* $\langle FE \cdot \sigma, A \cdot \sigma \rangle$ *of forced events and actions in a force rule* $fr =$ **force** FE **on** E **if** C **do** A *such that* $fr \in FrcRules$*,* $E \cdot \sigma' \subseteq \Xi$*,* $\mathbf{s}_l(\Delta, C, \sigma'')$ *holds,ignored*$(\Delta, \Xi, E, IgnRules)$ *does not hold and* $\sigma = \sigma' \cup \sigma''$*; Es is the largest set of forced events Ev such that* $\langle Ev, A \rangle \in EAs$*;* $\Xi' = \Xi \cup Es$*; As is the largest set of actions A such that* $\langle Ev, A \rangle \in EAs$*; and* $\mathbf{s}'_r(\Delta, IfRules, PrvRules, As, \Delta')$ *holds.*

Relation $\mathbf{s}^*$ calculates the new state of affairs Δ' from an initial state Δ and a set Ξ of events that occurred applying a list of ECA-rules, if-rules, ignore-rules, prevent-rules and force-rules.

Definition 13. *Relation* $\mathbf{s}^*(\Delta, \Xi, ECARules, IfRules, IgnRules, PrvRules, FrcRules, \Delta')$ *mapping a state of affairs* Δ*, a list* Ξ *of events that occurred, a list of ECA-rules, a list of if-rules, a list of ignore-rules, a list of prevent-rules, a list of force-rules and a new state of affairs holds iff Cs is the largest set of conditions C such that retract*$(fired(C))$ *holds; assert*$(fired(false))$*,* $\mathbf{s}_{if}(\Delta, IfRules, PrvRules, \Delta'')$*,* $\mathbf{s}_f(\Delta'', \Xi, IfRules, IgnRules, PrvRules, FrcRules, \Xi', \Delta''')$ *and* $\mathbf{s}_{on}(\Delta''', \Xi', ECARules, IfRules, IgnRules, PrvRules, \Delta')$ *hold.*

3 Example of Concurrency: Soup Bowl Lifting

In this section we present an example on how to use the $\mathcal{I}$ language in order to specify a variation of a problem about concurrent action: the Soup Bowl Lifting problem [9]. Picture a situation where a soup bowl has to be lifted by two (physical) agents; one lifting from the right-hand side and the other one from the left-hand side. If both sides are not lifted simultaneously then the soup spills.

The order in which the rules are declared is important since they are executed in the order they are declared. We do not obtain the same effect with rules 7, 8 and 9 (finally *spilled* does not hold after lifted from both sides simultaneously) than with rules 9, 7 and 8 (finally *spilled* holds even after lifted from both sides simultaneously).

$$\textbf{on } pushLeft \textbf{ if } true \textbf{ do } \oplus spilled \tag{7}$$

$$\textbf{on } pushRight \textbf{ if } true \textbf{ do } \oplus spilled \tag{8}$$

$$\textbf{on } pushLeft, pushRight \textbf{ if } true \textbf{ do } \ominus spilled \bullet \ominus onTable \tag{9}$$

Rules 7 and 8 specify that the soup is spilled whenever the bowl is lifted either from the right-hand side or the left-hand side. However, rule 9 removes the spill effect whenever both events are done simultaneously. However, with rules 9, 7 and 8, we do not obtain the desired result since the *spilled* formula may be added after executing the rule that removes *spilled* formula.

To prevent the bowl from spilling, we may add the next rule to rules 7-9:

$$\textbf{prevent } \textit{spilled} \textbf{ if } \textit{true} \tag{10}$$

However, adding the following rules instead would also prevent the bowl to be lifted since ignoring one event will prevent all the combined events to be considered.

$$\textbf{ignore } \textit{pushLeft} \textbf{ if } \textit{true} \tag{11}$$

$$\textbf{ignore } \textit{pushRight} \textbf{ if } \textit{true} \tag{12}$$

Contrarily, if we add rule 13 to rules 7-9, we prevent the bowl to be lifted from both sides simultaneously but not to be only lifted from one side since we are only ignoring the events if they occur together.

$$\textbf{ignore } \textit{pushLeft}, \textit{pushRight} \textbf{ if } \textit{true} \tag{13}$$

This basic example give us a sample of the expressiveness of $\mathcal{I}$. In the next section, we introduce electronic institutions and the meaning of the formulae needed for representing them in $\mathcal{I}$.

4 Electronic Institutions

Our work extends *electronic institutions* (EIs) [10][2], providing them with a normative layer specified in terms of ignore, prevent and force rules. There are two major features in EIs: the *states* and *illocutions* (*i.e.*, messages) uttered (*i.e.*, sent) by those agents taking part in the EI. The states are connected via edges labelled with the illocutions that ought to be sent at that particular point in the EI. Another important feature in EIs are the agents' *roles*: these are labels that allow agents with the same role to be treated collectively thus helping engineers abstract away from individuals. We define below the class of illocutions we aim at – these are a special kind of term:

Definition 14. *Illocutions* I *are terms* $ill(p, ag, r, ag', r', \tau)$ *where* p *is a performative (*e.g. inform *or* request*);* ag, ag' *are agent identifiers;* r, r' *are role labels; and* τ *is a term with the actual content of the message.*

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

An institutional state is a state of affairs that stores all utterances during the execution of a MAS, also keeping a record of the state of the environment, all observable attributes of agents and all the expectations associated with the agents.

We differentiate two kinds of events, with the following intuitive meanings:

1. I – an agent uttered illocution I.
2. $newtick(t)$ – a new tick of the clock occurred at time t.

[2] EI scenes are basically covered with ECA rules

We shall use event 2 above to obtain the time with which illocutions and expectations are time-stamped.

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

1. $inst(\mathsf{I}, t)$ – I was accepted as an institutional utterance at time t.
2. $exp(\overline{\mathsf{I}}, t)$ – $\overline{\mathsf{I}}$ is expected to be uttered since time t.

We allow agents to utter whatever they want (via I events). However, the unwanted utterances may be discarded and/or may cause sanctions, depending on the deontic notions we want or need to implement via our rules. The *inst* formulae are thus *confirmations* of the I events. We shall use formula 2 above to represent expectations of agents within EIs.

5 Applied Example: Bank

In this section we introduce an example of banking institution where agents are allowed to do certain operations with money. The operations in our bank are depositing, withdrawing and transferring. In our example we have two types of accounts called a and b owned by two different agents. In order to perform an operation in one of these accounts both agents have to *simultaneously* make the proper request.

Type a accounts have the limitation that no withdrawing, transferring from and debiting is allowed having a negative credit. If it is the case and there is enough money in a type b account of the same agent then necessary credit is automatically transferred to the account with negative credit and a fee is debited.

Type b accounts have the following limitations:

1. They cannot be in red. All the transactions that would finish in negative credit are rejected.
2. Withdrawing from or depositing to these accounts is not allowed.

Rule 14 specify the effects of opening an account of type T to agents $A1$ and $A2$ with an amount M of credit if another account of the same type with the same owners is not already opened.

$$\begin{array}{ll} \textbf{on} & newtick(Time), open\_account(Id, A1, A2, T, M) \\ \textbf{if} & \neg account(Id, A1, A2, T, \_) \wedge \neg account(Id, A2, A1, T, \_) \\ \textbf{do} & \oplus account(Id, A1, A2, T, M)\bullet \\ & \oplus inst(open\_account(Id, A1, A2, T, M), Time) \end{array} \tag{14}$$

Rule 15 specify the effect of withdrawing a given quantity M_q of money from a given account due to the simultaneous request of both owners of the account. The rules in the action section calculate the new credit for the account and modifies its value by removing the old credit and adding the new one. Likewise, a rule for the effects of depositing may also be specified.

$$\begin{array}{ll} \textbf{on} & newtick(Time), withdraw(A1, Id, M_q), withdraw(A2, Id, M_q) \\ \textbf{if} & account(Id, A1, A2, T, M) \\ \textbf{do} & M2 = M - M_q \bullet \ominus account(Id, A1, A2, T, M)\bullet \\ & \oplus account(Id, A1, A2, T, M2) \bullet \oplus inst(withdraw(A1, A2, Id, M_q), Time) \end{array} \tag{15}$$

Rule 16 specifies the effect of transferring from one account (of an agent and of a certain type) to another account possibly as payment of a certain concept C: the source account is reduced and the destination account is increased by the stated amount.

$$\begin{array}{ll} \textbf{on} & \mathit{newtick}(\mathit{Time}), \mathit{transfer}(A1, Id_s, Id_d, C, M), \mathit{transfer}(A2, Id_s, Id_d, C, M) \\ \textbf{if} & \mathit{account}(Id_s, A1, A2, T_s, M_s) \wedge \mathit{account}(Id_d, A3, T_d, M_d) \\ \textbf{do} & M2_s = M_s - M \bullet \ominus \mathit{account}(Id_s, A1, A2, T_s, M_s) \bullet \\ & \oplus \mathit{account}(Id_s, A1, A2, T_s, M2_s) \bullet M2_d = M_d + M \bullet \\ & \ominus \mathit{account}(Id_d, A3, T_d, M_d) \bullet \oplus \mathit{account}(Id_d, A3, T_d, M2_d) \bullet \\ & \oplus \mathit{inst}(\mathit{transfer}(A1, A2, Id_s, Id_d, C, M), \mathit{Time}) \end{array} \quad (16)$$

To avoid concurrent actions affecting the same account, we use rule 17. In this case, only the first action is taken into account and the rest of concurrent actions are ignored.

$$\textbf{prevent } \mathit{account}(I, A_1, A_2, T, M) \wedge \mathit{account}(I, A_1, A_2, T, M_2) \textbf{ if } M \neq M_2 \quad (17)$$

In our example, accounts of type a have the restriction that agents are not allowed to withdraw or transfer from a accounts with negative credit. This is achieved with rules like:

$$\textbf{ignore } \mathit{withdraw}(A, Id, \_) \textbf{ if } \mathit{account}(Id, A, \_, a, M) \wedge M < 0 \quad (18)$$

$$\textbf{ignore } \mathit{transfer}(A, Id_s, \_, \_, \_) \textbf{ if } \mathit{account}(Id_s, A, \_, a, M) \wedge M < 0 \quad (19)$$

Accounts of type b also have some restrictions. First, they cannot go into negative numbers. This is achieved with the following rule:

$$\textbf{prevent } \mathit{account}(Id, A1, A2, b, M) \textbf{ if } M < 0$$

Second, agents are not allowed to withdraw from accounts of type b. This is achieved by rule 20.

$$\textbf{ignore } \mathit{withdraw}(\_, Id, \_) \textbf{ if } \mathit{account}(Id, \_, \_, b, \_) \quad (20)$$

Furthermore, if an account of type a goes into the negatives then the necessary amount to avoid this situation is transferred from an account of type b. Rule 21 forces this type of events. Notice that a similar rule but with the order of the owners of the accounts reversed is also necessary since the owners may not appear in the same order.

$$\begin{array}{ll} \textbf{force} & \mathit{transfer}(A, Id_b, Id_a, a\_negative, C), \mathit{transfer}(A2, Id_b, Id_a, a\_negative, C) \\ \textbf{if} & \mathit{account}(Id_a, A, A2, a, C2) \wedge C2 < 0 \wedge C = C2 \wedge \\ & \mathit{account}(Id_b, A, A2, b, C3) \wedge C3 \geq C \end{array} \quad (21)$$

6 Related Work

In the model of Electronic Institutions of [10], agent interaction is brought about by uttering illocutions and it is decomposed in a set of scenes where only one illocution is accepted as legal simultaneously. As for norms, agents may be expected to utter certain illocutions under given conditions. However, there is no

notion of prevention of a state or force of events. Furthermore, only events that are not part of the protocol are ignored, not allowing to write further conditions in which an illocution is ignored.

The work presented in this paper is the result of the evolution of our previous work on norm languages for electronic institutions [1]. In that work, we presented a rule language that does not use forward chaining to calculate the effects of events and to explicitly manage normative positions (i.e. permissions, prohibitions and obligations). For the present work, we use those rules in the form of event-condition-action rules. Then, we added standard condition-action rules that use forward chaining. Furthermore, we changed our standard deontic notions that only regulated one illocution into more institutional-centred notions as ignoring, forcing or expecting concurrent events or preventing an institutional state.

$n\mathcal{C}_+$ is a language for representing and reasoning about action domains that include some normative notions [3]. Its semantics is based on labelled transition systems. The language allows to make queries about the transition system generated from an action description allowing to pre-dict, post-dict, or plan. In the normative aspect, $n\mathcal{C}_+$ only labels states and actions as green or red without including our notion of prevention that ignores actions that lead to an unwanted state. We can obtain this labeling by adding *green* to the initial state and rules of the form "**on** *events* **if** *conditions* **do** $\ominus$*green* $\bullet$ $\oplus$*red*" or "**if** *condition* **do** $\ominus$*green* $\bullet$ $\oplus$*red*". Instead of using ignore-rules, $n\mathcal{C}_+$ may label events as non-executable obtaining no solution when this kind of events occur. Since we want to maintain the state of the multi-agent system, we would need to ignore all the actions that occurred in that moment even the ones that does not lead to an unwanted state.

The implementation of $n\mathcal{C}_+$ loads the full transition system in order to resolve the queries. When dealing with fluents with large numeric values, the implementation suffers from a state explosion increasing the load and resolution time. As mentioned above, we are aiming at monitoring and maintaining the state of the enactment of open regulated multi-agent systems. To use the implementation of $n\mathcal{C}_+$ in this setting, we would have to add the new agents to the action description file and reload it again. However, the long time that elapses to complete this operation makes unviable the use of the implementation for our purposes and motivated this work.

7 Conclusions and Future Work

In this paper we have introduced a formalism for the management and regulation of concurrent events generated by agents in open MAS. Ours is a rule language in which concurrent events may have a combined effect and may be ignored, forced, expected or sanctioned. The semantics of our formalism relies on transition systems conferring it a well-studied semantics.

We have explored our proposal in this paper by specifying an example of concurrency: soup bowl lifting problem and an example of bank as Electronic Institution.

Although our language is not as expressive as the language of [3] since we cannot post-dict or plan about an action description, our language is not a language for checking properties of a transition system but for specifying its behaviour.

As a proof of concept, an interpreter of $\mathcal{I}$ were implemented in Prolog. As future work, we would like to embed this interpreter in a real MAS and include the distributed management of normative positions introduced in [11].

Acknowledgements – This work was partially funded by the Spanish Education and Science Ministry as part of the projects TIN2006-15662-C02-01 and 2006-5-0I-099 and it was partially done during a stay of the author in Imperial College London. The author wants to thank Marek Sergot for his advise and hospitality. He also thanks Juan-Antonio Rodríguez-Aguilar and Pablo Noriega for their comments and reviews. García-Camino enjoys an I3P grant from the Spanish National Research Council (CSIC).

References

1. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Norm Oriented Programming of Electronic Institutions. In: Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'06). (2006)
2. Artikis, A., Kamara, L., Pitt, J., Sergot, M.: A Protocol for Resource Sharing in Norm-Governed Ad Hoc Networks. In: Declarative Agent Languages and Technologies II. Volume 3476 of LNCS. Springer-Verlag (2005)
3. Sergot, M., Craven, R.: The deontic component of $n\mathcal{C}+$. In: Eighth International Workshop on Deontic Logic in Computer Science (DEON'06). (2006)
4. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., Torroni, P.: Mapping deontic operators to abductive expectations. In: Proceedings of 1st International Symposium on Normative Multiagent Systems (NorMAS 2005), AISB 2005, Hertfordshire, Hatfield, UK (2005)
5. Minsky, N.: Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual). Technical report, Rutgers University (2005)
6. von Wright, G.H.: Norm and Action: A Logical Inquiry. Routledge and Kegan Paul, London (1963)
7. Apt, K.R.: From Logic Programming to Prolog. Prentice-Hall, U.K. (1997)
8. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, U.S.A. (1990)
9. Gelfond, M., Lifschitz, V., Rabinov, A.: What are the limitations of the Situation Calculus? Essays in Honor of Woody Bledsoe (1991) 167–179
10. Esteva, M.: Electronic Institutions: from specification to development. PhD thesis, Universitat Politecnica de Catalunya (2003) Number 19 in IIIA Monograph Series.
11. Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A., Vasconcelos, W.: Distributed Norm Management in Regulated Multi-agent Systems. In: Sixth International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'07). (2007)