Constraint Rule-based Programming of Norms for Electronic Institutions

Andrés García-Camino[1], Juan-Antonio Rodríguez-Aguilar[1], Carles Sierra[1], and Wamberto Vasconcelos[2]

[1]IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Scientific Research Council
Campus UAB 08193 Bellaterra, Catalunya, Spain
{andres, jar, sierra}@iiia.csic.es
[2]Dept. of Computing Science, Univ. of Aberdeen,
AB24 3UE, Scotland, UK
wvasconcelos@acm.org

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

1 Introduction

A major challenge in multi-agent system (MAS) research is the design and implementation of *open* multi-agent systems in which coordination must be achieved among self-interested agents defined with different languages by several designers [23]. Norms can be used for this purpose as a means to regulate the observable behaviour of agents as they interact in pursuit of their goals [5,10,30,51]. There is a wealth of socio-philosophical and logic-theoretical literature on the subject of norms (*e.g.*, [40,42]). More recently, much attention has been paid to more pragmatic and implementational aspects of norms, that is, how norms can be given a computational interpretation and how norms can be factored in in the design and execution of MASs (*e.g.*, [4,9,15,16,34]).

Ideally, norms, once captured via some suitable formalism, should be directly executed, thus realising a computational, normative environment wherein agents interact. Computational norms are applicable when the current representation of the system complies with certain conditions. When representing agents from a social point of view, they are characterised by their observable attributes and normative position. A normative position [40] is the "social burden" associated

with individual agents, that is, their obligations, permissions and prohibitions. Depending on what agents do, their social representation (*i.e.*, the perception that other agents can have of them, that is, normative positions and observable attributes) may change – for instance, social reputation can increase, permissions/prohibitions can be revoked or obligations, once fulfilled, removed.

Norm-oriented programming is a programming paradigm aimed at equiping engineers with means to directly specify via norms how the interaction among the components of a MAS (*viz.*, the agents and their computational environment) should be regulated. This regulation can be done in many ways, including, for instance, directly via general purpose programming languages or agent-oriented programming languages such as AgentSpeak [11]. However, in this paper we advocate the explicit use of norms and normative positions to specify how open and heterogeneous MASs should be regulated. Norm-oriented programming should be seen as complementary to approaches to model the internal behaviour of the components of the system like agent-oriented programming [41] or client-server paradigm. An example of norm-oriented programming for the Internet would be a firewall that rejects or forwards messages following a set of rules. In this example, applications can be either permitted or forbidden to send several types of messages but since firewalls do not keep track of the obligations of applications, this example does not fully implement the norm-oriented paradigm.

We try to make headway along this direction by introducing an executable language to specify agents' *normative positions* and manage their changes as agents interact via speech acts [39]. Our language works as a "machine language" for norms on top of which higher-level normative languages can be accommodated.

This language has been conceived to represent distinct flavours of deontic notions and relationships: we can define different normative contexts in which different deontic notions hold. In our language, we can specify two concurrent normative contexts such that in one of them prohibitions cannot be violated and in the other one prohibitions over certain actions can be violated under penalties.

Our language is rule-based and we achieve greater flexibility, expressiveness and precision than conventional production systems by allowing constraints [21,33] over variables to appear in our constructs. Constraints are first-class entities managed explicitly – we accommodate, as we show, constraints in our semantics using standard constraint solving techniques. Constraints allow for more sophisticated notions of norms and normative positions to be expressed. For instance, in a scenario in which a selling agent is obliged to deliver a product satisfying some quality requirements before a deadline, both the quality requirements and the delivery deadline can be regarded as constraints that must be considered as part of the agent's obligation. Thus, when the agent delivers the good satisfying all the constraints, we should regard the obligation as fulfilled. Notice too that since the deadline might eventually be changed, we also require the capability of modifying constraints at run-time.

One of the first models of open MAS that regulates the interaction among agents without assuming any internal feature of the agents are electronic institu-

tions (EIs) [35,38,12]. Despite being successful in achieving a significant degree of openness, electronic institutions are strict in the sense that only those interactions which are part of the design can take place. Our normative approach gives more flexibility to EIs in that we can also capture deviant behaviour. Our work sets the foundations to specify and implement open regulated MASs via norms.

The structure of this paper is as follows. In the next section we present desirable properties of normative languages. In section 3 we describe the syntax and semantics of our normative language – we explain, in various sections, how our language addresses all these requirements. Section 4 summarises electronic institutions and explains how we capture normative positions of participating agents. We put our language to use in sections 5.1 and 5.2 where, respectively, we define institutional states and rules. We illustrate the usefulness of our language with a specification of the Dutch Auction protocol in section 5.4. In section 6 we show how our language captures a sample of other contemporary approaches and in section 7 we compare our approach with other related work. Finally, we draw conclusions and outline future work in section 8.

2 Desiderata for Norm-Oriented Languages

We aim at a language to support the specification of coordination mechanisms in MASs via norms. For this purpose, we identify and justify the desirable features we expect in candidate languages:

- **Explicit management of normative positions** – We take the stance that we cannot refer to agents' mentalistic notions, but only to their observable actions and their normative positions. Notice that as a result of agents' observable, social interactions, their normative positions [40] change. Hence, the first requirement of our language is to support the *explicit management* of agents' normative positions.
- **General purpose** – Turning our attention to theoretical models of norms, we notice that there is a plethora of deontic logics with different axioms to establish relationships among deontic notions. Thus, we require that our language captures different deontic notions along with their relationships. In other words, the language must be of *general purpose* so that it helps MAS designers to specify the widest range of normative systems as possible.
- **Pragmatic** – In a sense, we pursue a "machine language" for norms on top of which alternative higher-level languages can be accommodated. Along this direction, and from a language designer's point of view, it is fundamental to identify the *norm patterns* (*e.g.*, conditional obligation, time-based permissions and prohibitions, continuous obligation, and so on) in the literature and ensure that the language supports their encoding. In this way, not only shall we be guaranteeing the expressiveness of our language, but also addressing pragmatic concerns by providing *design patterns* to guide and ease MAS design.
- **Declarative** – In order to ease MAS programming, we shall also require our language to be *declarative*, with an implicit execution mechanism to reduce

the number of issues designers ought to concentrate on. As an additional benefit, we expect its declarative nature to facilitate verification of properties of the specifications.

3 A Rule-based Language for Managing Normative Positions

In this section we introduce a rule-based language for the explicit management of events generated by agents and the effects they cause [17,18,19,20]. We consider that agents can (directly or indirectly) cause changes in their own normative positions (*e.g.*, by bidding in an auction), in the normative positions of other agents (*e.g.*, by delegating or commanding), in the observable attributes of agents (*e.g.*, "badmouthing" an agent can decrease its reputation), or in the state of resources of the environment (*e.g.*, moving a box changes its location). By *environment* we mean the shared resources which are not part of the agents and, therefore, cannot be freely accessed and modified. By *state of affairs* we mean the representation of aspects of the MASs enactment including the set of attributes that a community of agents can access or modify in an unregulated setting.

In regulated MASs these attributes can only be accessed and modified under certain conditions. Our rule-based language allows us to represent regulated changes in an elegant way and also fulfils the requirement that a normative language should be declarative. The rules depict how the state of affairs changes as agents interact with each other or the environment.

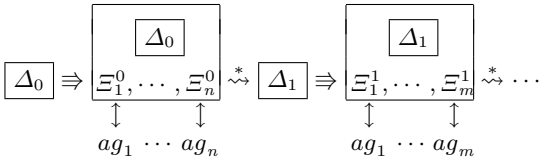

Fig. 1. Semantics as a Sequence of Δ's

Figure 1 depicts the computational model we propose. An initial state of affairs Δ_0 (possibly empty) is offered (represented by "$\Rightarrow$") to a set of agents $(ag_1, \cdots, ag_n)$. These agents can add their events $(\Xi_1^0, \cdots, \Xi_n^0)$ to the state of affairs (via "$\updownarrow$"). Ξ_i^t is the (possibly empty) set of events added by agent i at state of affairs Δ_t and an event is a special case of atomic formula. After a established amount of time, we perform an exhaustive application of rules (denoted by "$\overset{*}{\rightsquigarrow}$") to the modified state, yielding a new state of affairs Δ_1. This new state will, on its turn, be offered to the agents for them to add their events, and the same process will go on.

Although our language can be used for regulated MAS in general, in this paper we concentrate on a specific model, the so called *electronic institutions* (cf. section 4). The events in that model are speech acts and time events only, so we shall focus on these in the rest of this paper.

3.1 Preliminary Definitions

We initially define some basic concepts. The building blocks of our language are *terms*:

Definition 1. *A term, denoted as τ, is*

- *Any variable x, y, z (with or without subscripts) or*
- *Any construct $f^n(\tau_1, \ldots, \tau_n)$, where f^n is an n-ary function symbol and $\tau_1, \ldots, \tau_n$ are terms.*

Terms f^0 stand for *constants* and will be denoted as a, b, c (with or without subscripts). We shall also make use of numbers and arithmetic functions to build our terms; arithmetic functions may appear infix, following their usual conventions. We adopt Prolog's convention [3] using strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants. Some examples of terms are $Price$ (a variable) and $send(a, B, Price \times 1.2)$ (a function). We also need to define *atomic formulae*:

Definition 2. *An atomic formula, denoted as α, is any construct $p^n(\tau_1, \ldots, \tau_n)$, where p^n is an n-ary predicate symbol and $\tau_1, \ldots, \tau_n$ are terms.*

When the context makes it clear what n is we can drop it. p^0 stands for propositions. We shall employ arithmetic relations (*e.g.*, $=$, $\neq$, and so on) as predicate symbols, and these will appear in their usual infix notation. We also make use of atomic formulae built with arithmetic relations to represent *constraints* on variables – these atomic formulae have a special status, as we explain below. We give a definition of our constraints, a subset of atomic formulae:

Definition 3. *An arithmetical constraint γ is a binary atomic formula in the infix form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.*

In the rest of the paper we will use constraints and arithmetical constraints indistinctly. We need to differentiate ordinary atomic formulae from constraints. We shall use $\bar{\alpha}$ to denote atomic formulae that are *not* constraints.

A state of affairs is a set of atomic formulae, representing (as shown below) the normative positions of agents, observable agent attributes and the state of the environment[1].

Definition 4. *A state of affairs $\Delta = \{\alpha_0, \ldots, \alpha_n\}$ is a a finite and possibly empty set of implicitly, universally quantified atomic formulae $\alpha_i, 0 \leq i \leq n$.*

[1] We refer to the *state of the environment* as the subset of atomic formulae representing observable aspects of the environment in a given point in time.

3.2 A Language for Rules with Constraints

Our rules are constructs of the form $LHS \rightsquigarrow RHS$, where LHS contains a representation of parts of the current state of affairs which, if they hold, will cause the rule to be triggered. RHS describes the updates to the current state of affairs, yielding the next state of affairs:

Definition 5. *A rule, denoted as* R*, is defined as:*

$$\begin{aligned} \mathsf{R} &::= LHS \rightsquigarrow RHS \\ LHS &::= LHS \wedge LHS \mid LHS \vee LHS \mid \neg LHS \mid \mathsf{Lit} \\ RHS &::= \mathsf{U} \bullet RHS \mid \mathsf{U} \\ \mathsf{Lit} &::= \alpha \mid \mathtt{sat}(\Gamma) \mid x = \{\alpha \mid LHS\} \\ \mathsf{U} &::= \oplus\alpha \mid \ominus\alpha \end{aligned}$$

where x is a variable name.

Intuitively, the left-hand side LHS describes the conditions the current state of affairs oughts to have for the rule to apply. The right-hand side RHS describes the updates to the current state of affairs, yielding the next state of affairs.

In the next section we define the semantics of each construct above, but informally, the construct α checks whether the atomic formulae α is in the state of affairs, $\mathtt{sat}(\Gamma)$ checks whether Γ (a set of constraints) is satisfied in the state of affairs. We also make use of a special kind of term, called a *set constructor*, represented as $\{\alpha \mid LHS\}$. This construct is useful when we need to refer to all atomic formulae in the state of affairs (αs) for which LHS holds. For instance, $\{p(A,B,C) \mid \mathtt{sat}(B > 20) \wedge \mathtt{sat}(C < 100)\}$ stands for the set of atomic formulae $p(A,B,C)$ such that B is greater than 20 and C is less than 100. Notice that $\{p(A,B,C) \mid B > 20 \wedge C < 100\}$ stands for the set of atomic formulae $p(A,B,C)$ with at least these two constraints associated: B is constrained to be greater than 20 and C is constrained to be less than 100. That is, it checks whether both constraints are in the state of affairs. The Us represent updates: they add to the state of affairs (via operator $\oplus$) or remove from the state of affairs (via operator $\ominus$) atomic formulae.

3.3 Semantics of Rules

As shown in figure 1, we define the semantics of our rules as a relationship between states of affairs: rules map an existing state of affairs to a new state of affairs. In this section we define this relationship. Initially we need to refer to the set of constraints of a state of affairs. We call $\Gamma = \{\gamma_0, \ldots, \gamma_n\}$ the set of all constraints in Δ, and formally relate a state of affairs to its constraints as follows:

Definition 6. *Given a state of affairs Δ, relationship $constrs(\Delta, \Gamma)$ holds iff Γ is the smallest set such that for every constraint $\gamma \in \Delta$ then $\gamma \in \Gamma$.*

In the definitions below we rely on the concept of *substitution*, that is, the set of values for variables in a computation [3,13]:

Definition 7. *A substitution* $\sigma = \{x_0/\tau_0, \ldots, x_n/\tau_n\}$ *is a finite and possibly empty set of pairs* x_i/τ_i, $0 \leq i \leq n$.

Definition 8. *The application of a substitution to an atomic formulae* α *is as follows:*

1. $c \cdot \sigma = c$ *for a constant* c;
2. $x \cdot \sigma = \tau \cdot \sigma$ *if* $x/\tau \in \sigma$; *otherwise* $x \cdot \sigma = x$;
3. $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.

Definition 9. *The application of a substitution to a sequence is the sequence of the application of the substitution to each element:* $\langle \alpha_1, \ldots, \alpha_n \rangle \cdot \sigma = \langle \alpha_1 \cdot \sigma, \ldots, \alpha_n \cdot \sigma \rangle$

We now define the semantics of the LHS of a rule, that is, how a rule is triggered:

Definition 10. $\mathbf{s}_l(\Delta, LHS, \sigma)$ *holds between state* Δ, *the left-hand side of a rule* LHS *and a substitution* σ *depending on the format of* LHS:

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

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. Cases 3 holds when an atomic formulae (a predicate or constraint) is part of the state of affairs. Case 4 holds if the set of constraints on the LHS added to the constraints in the state of affairs (Γ') are satisfiable. The substitution σ obtained so far is applied to γ. It uses the predicate $satisfiable/1$, that checks whether a set of constraints is satisfiable.[2] Case 5 specifies the semantics for *set constructors*: x is the set of atomic formulae that satisfy the conditions of the set constructor.

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

Definition 11. *Relation* $\mathbf{s}_r(\Delta, RHS, \Delta')$ *mapping a state* Δ, *the right-hand side of a rule* RHS *and a new state* Δ' *is defined as:*

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

[2] Our work builds on standard technologies for constraint solving – in particular, we have been experimenting with SICStus Prolog [44] constraint satisfaction libraries [22].

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 caters for the insertion of atomic formulae $\bar{\alpha}$ which do not conform to the syntax of constraints. Case 3 defines how a constraint is added to a state Δ: the new constraint is checked whether it can be satisfied with the existing constraints Γ and then it is added to Δ'. Case 4 caters for the removal of atomic formulae (both constraints and non-constraints).

To complete the definition of the rule system, we define the semantics of our rules as relationships between states of affairs: rules map an existing state of affairs to a new state of affairs, thus modelling transitions between states of affairs. We adopt the usual semantics of production rules [27], that is, we exhaustively apply each rule by matching its LHS against the current state of affairs and use the values of variables obtained in this match to instantiate RHS via $\mathbf{s}_r$.

3.4 An Interpreter for Rules with Constraints

The semantics above provide a basis for the implementation of a rule interpreter. Although we have implemented it with SICStus Prolog [44] we show how a rule is interpreted in figure 2 as a logic program, interspersed with built-in Prolog predicates; for easy referencing, we show each clause with a number on its left.

1. $\mathbf{s}_l(\Delta, (LHS \wedge LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma'), \mathbf{s}_l(\Delta, LHS' \cdot \sigma', \sigma''), \sigma = \sigma' \cup \sigma''$
2. $\mathbf{s}_l(\Delta, \neg\, LHS, \sigma) \leftarrow \neg\, \mathbf{s}_l(\Delta, LHS, \sigma)$
3. $\mathbf{s}_l(\Delta, \alpha, \sigma) \leftarrow \texttt{member}(\alpha \cdot \sigma, \Delta)$
4. $\mathbf{s}_l(\Delta, \texttt{sat}(\Gamma), \sigma) \leftarrow constrs(\Delta, \Gamma'), append(\Gamma, \Gamma', \Gamma''), satisfiable(\Gamma'' \cdot \sigma)$
5. $\mathbf{s}_l(\Delta, x = \{\alpha \mid LHS\}, \{x/AllAtfs\}) \leftarrow \texttt{findall}(\alpha \cdot \sigma, \mathbf{s}_l(\Delta, \alpha \wedge LHS, \sigma), AllAtfs)$
6. $\mathbf{s}_r(\Delta, (\mathsf{U} \bullet RHS), \Delta'') \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta'), \mathbf{s}_r(\Delta', RHS, \Delta'')$
7. $\mathbf{s}_r(\Delta, \oplus\bar{\alpha}, \Delta') \leftarrow \Delta' = \{\bar{\alpha}\} \cup \Delta$
8. $\mathbf{s}_r(\Delta, \oplus\gamma, \Delta') \leftarrow constrs(\Delta, \Gamma), satisfiable(\{\gamma\} \cup \Gamma), \Delta' = \{\gamma\} \cup \Delta$
9. $\mathbf{s}_r(\Delta, \ominus\alpha, \Delta') \leftarrow \texttt{delete}(\Delta, \alpha, \Delta')$

Fig. 2. Interpreter for Rules with Constraints

For each rule, we apply $\mathbf{s}_l(\Delta, LHS, \sigma)$ and $\mathbf{s}_r(\Delta, RHS \cdot \sigma, \Delta')$ sequentially for all the different substitutions σ in the state of affairs such that $\mathbf{s}_l(\Delta, LHS, \sigma)$ holds. Clauses 1-5 and 6-12 are, respectively, adaptations of the cases depicted in Def. 10 and Def. 11.

We can define $satisfiable/2$ via the built-in `call_residue`/2 predicate, available in SICStus Prolog:

$$satisfiable(\{\gamma_1, \ldots, \gamma_n\}) \leftarrow \texttt{call\_residue}((\gamma_1, \ldots, \gamma_n), \_)$$

It is worth mentioning that in the actual Prolog implementation, substitutions σ appear implicitly as values of variables in terms – the logic program above will look neater (albeit farther away from the definitions) when we incorporate this.

3.5 Pragmatics of Rules with Constraints

In this section we illustrate the pragmatics of our rules with some examples:

$$(event(a,\, T) \wedge p(X)) \rightsquigarrow (\ominus event(a,\, T) \bullet \ominus p(X) \bullet \oplus p(X+1)) \quad (1)$$

$$(event(b,\, T) \wedge (T < D)) \rightsquigarrow \begin{pmatrix} \ominus event(b,\, T) \bullet \\ \ominus (T < D) \bullet \oplus (T < (D + 10)) \end{pmatrix} \quad (2)$$

$$(event(c,\, T) \wedge q(D) \wedge \mathtt{sat}(T > D)) \rightsquigarrow (\ominus event(c,\, T) \bullet \oplus s(c,\, T)) \quad (3)$$

The first example shows an event a which occurred at time T depicting the circumstances in which the rule should be applied; in this case, we check that the formula $p(X)$ exists and, on the right-hand side we ensure the event is "consumed" (thus not triggering off the rule indefinitely) and $p(X)$ is updated to $p(X+1)$ – we implement a counter for events of type a. Example 2 illustrates the management of constraints: these can be manipulated like ordinary predicates. In that example, we show how events of type b can have constraints associated to the time they occur updated to the previous limit plus ten units of time. Again, we ensure the event is removed from the state of affairs to prevent the rule from being used indefinitely. Example 3 illustrates how constraints can additionally be *checked* for their satisfaction: when an event of type c occurs and there is a formula $q(D)$ (storing a deadline) and the time the event occurred is greater than D (this being checked as a constraint), then we remove the event and add a record of this situation to the state of affairs.

Our rules manage states of affairs, adding or removing formulae (expressed on the RHS) when certain conditions (expressed on the LHS) hold. As illustrated in figure 1, our approach accommodates the participation of agents: they add atomic formulae onto the current state of affairs – these formulae represent agent-related events, represented above as $event(E,\, T)$ that, together with further elaboration on the circumstances, will trigger off rules to update the state of affairs. Some synchronisation is required in this activity, as we cater for the agents to concurrently update a shared data structure – a simple synchronisation mechanism is explained in [17].

There are further concerns to be taken into account when designing rules. Clearly, what we choose to go in the state of affairs has an immediate influence as to what should appear in rules. Another concern is how we choose to represent events generated by agents. We show in this paper a representation proposal that includes information on who caused the event, the time, and a suitable description of the event.

4 Electronic Institutions

Human societies deploy institutions [36] to establish how interactions must be structured within an organization. Institutions represent the "rules of the game" in a society, including any (formal or informal) constraints devised to shape human interaction. Institutions are the framework within which human interaction

takes place, defining what individuals are obliged, forbidden and permitted to do and under what conditions. Furthermore, human institutions not only structure human interactions but also enforce individual and social behaviour by obliging all to act according to the norms.

Electronic institutions (EIs) [35,38,12] are the electronic counterpart of human institutions – they establish the expected behaviour of agent societies. An EI defines a set of constraints that articulate agent interactions, defining what they are permitted to do. An EI defines a regulated environment where heterogeneous (human and software) agents can participate by playing different roles and can interact by means of speech acts [39].

In this section we introduce electronic institutions as defined in [12]. We implement them in section 5, enriching them with further deontic notions and relationships among them. Due to space restrictions we cannot provide here a complete introduction to electronic institutions – we refer readers to [12] for a comprehensive description. However, to make this work self-contained we have to explain concepts we make use of later on. Although our discussion is focused on EIs it can be generalised to various other formalisms that share some basic features.

In EIs interaction is regulated by means of multi-agent protocols which have two major features – these are the *states* in a protocol and *illocutions* (*i.e.*, messages) uttered (*i.e.*, sent) by those agents taking part in the protocol. The states are connected via edges labelled with the illocutions that ought to be sent at that particular point in the protocol. 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 programmers abstract away from individuals. We define below the class of illocutions we aim at – these are a special kind of atomic formulae:

Definition 12. *Illocutions* I *are ground atomic formulae* $p(ag, r, ag', r', \tau, t)$ *where*

- p *is an element of a set of illocutionary particles (*e.g.*,* inform*,* request*, etc.).*
- ag, ag' *are agent identifiers.*
- r, r' *are role labels.*
- τ*, an arbitrary ground term, is the actual content of the message, built from a shared content language.*
- $t \in \mathbb{N}$ *is a time stamp.*

The intuitive meaning of $p(ag, r, ag', r', \tau, t)$ is that agent ag playing role r sent message τ to agent ag' playing role r' at time t. An example of an illocution is $inform(ag_4, seller, ag_3, buyer, offer(car, 1200), 10))$. Sometimes it is useful to refer to illocutions that are not fully ground, that is, they may have uninstantiated (free) *variables* within themselves – in the description of a protocol, for instance, the precise values of the message exchanged can be left unspecified. During the enactment of the protocol agents will produce the actual values which will give rise to a (ground) illocution. We can thus define *illocution schemes*:

Definition 13. *An illocution scheme* $\bar{\mathsf{I}}$ *is any atomic formula* $p(ag, r, ag', r', \tau, t)$ *whose terms are either variables or may contain variables.*

Another important concept in EIs we employ here is that of a *scene*. Scenes are self-contained sub-protocols with an initial state where the interaction starts and a final state where all interaction ceases. Scenes offer means to break down larger protocols into smaller ones with specific purposes.

For instance, we can have a registration scene where agents arrive and register themselves with an administrative agent; an auction scene depicts the interactions among agents wanting to buy and sell goods; a payment scene depicts how those agents who bought something in the auction scene ought to pay those agents they bought from. 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 I leads to another state. Different formalisms and approaches to protocol specification can be accommodated in our proposal, provided protocols can be broken down into uniquely defined states connected by edges; the edges are labelled with messages agents must send for the protocol to progress. Broadly speaking, an EI is specified as a set of scenes connected by transitions; these are points where agents may synchronise their movements between scenes [12].

Although all illocutions of a protocol are *permitted* some of them may be deemed *inappropriate* in certain circumstances. For instance, although a protocol may contemplate agents leaving the payment scene, it may be inappropriate to do so if the agent has not yet paid what it owes. Our rules further restrict the expected behaviour of agents, prohibiting them from uttering an illocution or adding constraints on the values of variables of illocutions. Rules can be triggered off by events involving any number of agents and their effects must persist until they are fulfilled or retracted by another rule.

5 Norm-Oriented Programming of Electronic Institutions

Despite successfully achieving a significant degree of openness, electronic institutions are strict in the sense that only explicitly permitted interactions can take place. As an initial step we pursue to implement rule-based electronic institutions in which deontic notions are not limited to the existent in the previous model of electronic institution.

We advocate a *separation of concerns*: rather than embedding normative aspects into the agents' design (say, by explicitly encoding normative aspects in the agent's behaviour) or coordination mechanisms (say, by addressing exceptions and deviant behaviour in the mechanism itself), we adopt the view that a coordination mechanism should be *supplemented* by an explicit and separate set of norms that further regulates the behaviour of agents as they take part in the enactment of a mechanism.

The separation of concerns should facilitate the design of MASs – as systems become more sophisticated, it becomes harder for engineers to address all the relevant features. By differentiating kinds of features and exploring them independently, engineers can "disentangle" them. However, the different components (coordination mechanisms and norms) must come together at some point in the

design process. In our view, norms further restrict the set of behaviours specified by the coordination mechanisms; a coordination mechanism, on its turn, determines if a set of norms can be fulfilled by those agents enacting it. Norms should be studied against their associated coordination mechanism and vice-versa.

In this section we use the language introduced in section 3 to program electronic institutions based on the notions introduced in section 4. In subsection 5.1 we specify how a state of affairs is represented in an EI, whereas in subsection 5.2 we make explicit the rules to transform such state of affairs at run-time.

5.1 Institutional States

An institutional state is a state of affairs that stores all utterances during the execution of a MAS, also keeping a record of the state of the environment, all observable attributes of agents and all obligations, permissions and prohibitions associated with the agents that constitute their normative positions.

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

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

We differentiate between utterances that are attempted to be accepted (*att*) and accepted utterances (*utt*). Since we aim at heterogeneous agents whose behaviour we cannot guarantee, Since we cannot guarantee agents' behaviour 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 rules – the illegal utterances may be discarded and/or may cause sanctions, depending on the deontic notions we want or need to implement. The *utt* formulae are thus *confirmations* of the *att* formulae.

We only allow fully ground attributes, illocutions and state control formulae (cases 1-4 above) to be present however, in formulae 6–8 s and w may be variables and $\bar{\mathsf{I}}$ may contain variables. We shall use formulae 4 to represent state change in a scene in relationship with global time passing. We shall use formulae 6–8 above to represent normative positions of agents within EIs.

We do not "hardwire"deontic notions in our semantics: the predicates above represent deontic operators but not their relationships. These are captured with rules as we show in section 5.2. We show in figure 3 a sample institutional state. The utterances show a portion 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

$$\Delta = \left\{ \begin{array}{l} utt(auction, w_2, inform(ag_4, seller, ag_3, buyer, offer(car, 1200), 10)), \\ utt(auction, w_3, inform(ag_3, buyer, ag_4, seller, buy(car, 1200), 13)), \\ obl(payment, w_4, inform(ag_3, payer, ag_4, payee, pay(Price), T_1)), \\ prh(payment, w_2, ask(ag_3, payer, X, adm, leave, T_2)) \\ oav(ag_3, credit, 3000), oav(car, price, 1200), \\ 1200 \leq Price, 20 > T_1 \end{array} \right\}$$

Fig. 3. Sample Institutional State

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 at least 1200 to agent ag_4 when the agents move to the payment scene; 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 restrict the values for *Price*, that is, the minimum value for the payment, and the latest time T_1 ag_3 is obliged to pay.

5.2 Institutional Rules

In this section we illustrate how expressive and flexible our rules are, yet they offer precision and ease-of-use. With the following examples we want to illustrate the generality of our language as required in section 2. Furthermore, we also provide some guidelines on how to specify the rules to update institutional states. Henceforth we shall call such rules *institutional rules*.

Providing Semantics to Deontic Notions We now provide some examples on how we explicitly manage normative positions of agents in our language as required in section 2. When specifying a normative system we need to define relationships among deontic notions. Such relationships should capture the pragmatics of normative aspects – what exactly these concepts mean in terms of agents' behaviour. 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, conferring on our approach the generality requirement in section 2. 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.

We can confer different degrees of enforcement on EIs . We start by looking at those illocutions that agents utter, *i.e.*, $att(S, W, I)$; these may become legal utterances, *i.e.*, $utt(S, W, I)$, if they are *permitted*, as specified by the following rule:

$$att(S, W, I) \wedge per(S, W, I) \rightsquigarrow \oplus utt(S, W, I) \tag{4}$$

That is, permitted attempts at utterances become legal utterances.

Attempts and prohibitions can be related together by the institutional rule:

$$att(S, W, I) \wedge prh(S, W, I) \rightsquigarrow sanction \tag{5}$$

Where *sanction* stands for atomic formulae representing sanctions on the agent who uttered a prohibited illocution. For instance, if the agent's credit is represented via $oav(Ag, credit, Value)$, the following rule applies a 10% fine on those agents who utter a prohibited illocution:

$$\begin{pmatrix} att(S, W, P(A_1, R_1, A_2, R_2, M, T)) \wedge \\ prh(S, W, P(A_1, R_1, A_2, R_2, M, T)) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ominus oav(A_1, credit, C) \bullet \\ \oplus oav(A_1, credit, C - C/10) \end{pmatrix} \tag{6}$$

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

$$\big(att(S, W, I) \wedge per(S, W, I) \wedge \neg\, prh(S, W, I) \big) \rightsquigarrow \oplus utt(S, W, I) \tag{7}$$

The rule above states that an utterance is accepted as legal whenever it is permitted and it is not the case that it is forbidden.

We can allow agents to do certain illegal actions (under harsher penalties if required):

$$\begin{array}{l} \begin{pmatrix} att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \wedge \\ \mathtt{sat}(Ag_1 \neq Ag_2) \ \wedge \ \mathtt{sat}(Ag_1 \neq Ag_3) \ \wedge \ \mathtt{sat}(Ag_2 \neq Ag_3) \end{pmatrix} \\ \rightsquigarrow \\ \begin{pmatrix} \ominus att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \bullet \\ \oplus utt(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T)) \end{pmatrix} \end{array} \tag{8}$$

The rule above states that if an agent attempts to reveal to Ag_2 (private) information about agent Ag_3, it is accepted without taking into account if it is forbidden or not. In both cases (rules 7 and 8), we can punish agents that violate prohibitions as shown in rule 6.

Dealing with inconsistency We can also capture further relationships among normative aspects and establish policies to cope with inconsistencies. For instance, we need to specify how to cope with the situation when an illocution is simultaneously obliged and forbidden – this may occur when an obligation assigned to agents in general (or to any agents playing a role) is revoked for a particular subgroup of agents or an individual agent (for instance, due to a sanction). In this case, we can choose to ignore/override either the obligation or the prohibition. For instance, without writing any extra rule we override the obligation and ignore the attempt to fulfil the obligation. The rule below ignores the prohibition and transforms an attempt to utter the illocution I into its utterance:

$$att(S, W, I) \wedge obl(S, W, I) \wedge prh(S, W, I) \rightsquigarrow \oplus utt(S, W, I) \tag{9}$$

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:

$$att(S, W, I) \wedge obl(S, W, I) \wedge prh(S, W, I) \rightsquigarrow \oplus exception(S, W, I) \qquad (10)$$

These examples illustrate how we explicitly manage normative positions of agents in our language as required in section 2.

5.3 Representing and Enacting Protocols via Institutional Rules

In the rest of the paper we consider scenes, presented in section 4, as the representation of protocols in EIs. The purpose of this section is to represent and build a computational model of the dynamics of an EI enactment, that is, its execution with our rule-based language. We concentrate our attention on EIs [12] (see section 4 above) but our approach addresses any protocol specified via non-deterministic finite-state machines.

We shall represent EIs declaratively as logic programs, as described in [47]. Each edge connecting two states of a scene will be denoted as the fact $edge(Scene, State, IllocutionScheme, NewState)$, representing that if the control of the enactment of *Scene* is in *State* and *IllocutionScheme* is uttered, then the control should move to *NewState*. Edges are compact descriptions of what can be said, *i.e.*, the permitted illocutions, and how the control of the enactment of the scene (and by extension, of the EI as a whole) should change as illocutions are uttered.

Protocols are permissions of what can be uttered and when it can be uttered. When permissions are combined with unconfirmed utterances (*i.e*, $att(s, w, \bar{\mathsf{I}})$, as captured by formula 4 above) and legal illocutions (*i.e*, $utt(s, w, \bar{\mathsf{I}})$) are combined with updates on the state of the enactment, then the protocol can be fully captured. In order to represent the control of the protocol enactment we use the term $ctr(Scene, State, TimeStamp)$, stored in the institutional state, which informs that at time *TimeStamp* the interaction enacted in *Scene* is at *State*.

The dynamics of the control of the enactment can be captured generically as the following institutional rule:

$$\begin{pmatrix} ctr(S, W_i, T) \\ att(S, W_i, I) \ \wedge \ per(S, W_i, I) \wedge \\ \neg\, prh(S, W_i, I) \ \wedge \ edge(S, W_i, I, W_j) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ominus ctr(S, W_i, T) \bullet \\ \oplus old\_ctr(S, W_i, T) \bullet \\ \oplus ctr(S, W_j, T+1) \bullet \\ \oplus utt(S, W_i, I) \end{pmatrix} \qquad (11)$$

That is, if the control of the enactment of scene S is now at state W_i and illocution I has been uttered and there is an edge connecting W_i with W_j labelled with that illocution, then the control of the enactment at the next time will move to $ctr(S, W_j, T{+}1)$. We keep track of the time of previous states using the *old_ctr* predicate.

The permissions of an agent society can be managed in various different manners. A simple and efficient way is to have permissions unchanged in the institutional state, that is, they are passed on from state to state without ever

being removed. Constraints, however, can be added to the variables of obligations as a result of the interactions among the agents.

We notice that institutional rules are expressive enough to represent normative aspects as well as institutional protocols (*i.e.*, scenes) and their enactment. Thus, we can claim that institutional rules addresses the requirements (introduced in section 2) of explicit management of normative positions and being general purpose and declarative.

5.4 Example: The Dutch Auction Protocol

In this section, we illustrate the pragmatics of our norm-oriented language by specifying the auction protocol employed in the fish market described in [35]. Following [35], the fish market can be described as a place where several scenes [12] take place simultaneously, at different locations, but with some causal continuity. The principal scene is the auction itself, in which buyers bid for boxes of fish that are presented by an auctioneer who calls prices in descending order, the so-called *downward bidding protocol*, a variation of the traditional Dutch auction protocol that proceeds as follows:

1. The auctioneer chooses a good out of a lot of goods that is sorted according to the order in which sellers deliver their goods to the sellers' admitter.
2. With a chosen good , the auctioneer opens a *bidding round* by quoting offers downward from the good's starting price, previously fixed by a sellers' admitter, as long as these price quotations are above a *reserve price* previously defined by the seller.
3. For each price the auctioneer calls, several situations might arise during the open round described below.
4. The first three steps are repeated until there are no more goods left.

The situations arising in step 3 are:

Multiple bids – Several buyers submit their bids at the current price. In this case, a collision comes about, the good is not sold to any buyer, and the auctioneer restarts the round at a higher price;

One bid – Only one buyer submits a bid at the current price. The good is sold to this buyer whenever his credit can support his bid. Otherwise, the round is restarted by the auctioneer at a higher price, the unsuccessful bidder is fined;

No bids – No buyer submits a bid at the current price. If the reserve price has not been reached yet, the auctioneer quotes a new price obtained by decreasing the current price according to the price step. Otherwise, the auctioneer declares the good as *withdrawn* and closes the round.

Proposed Solution Figure 4 shows the proposed protocol. As we proposed in section 5.3 the protocols are represented as a set of formula of the type $edge(S, W_i, I, W_j)$ and the rule 11. The situations arising in step 3 are captured in equations 12 – 17. For formatting reasons, we will use α_i to denote atomic formulae:

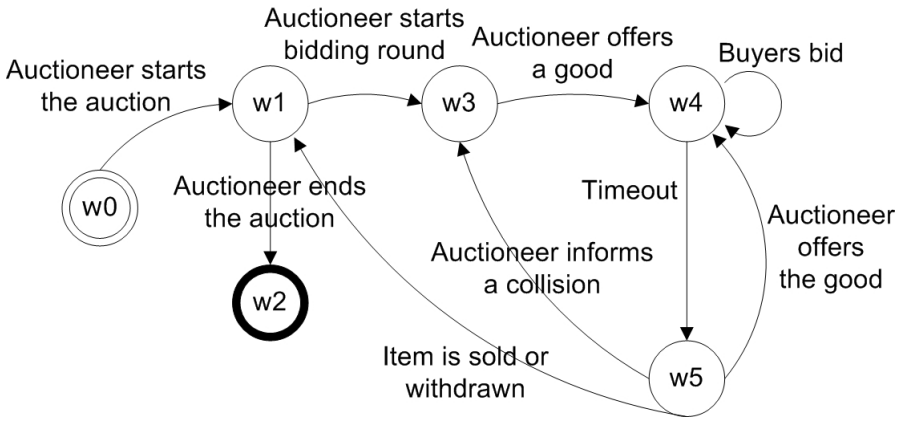

Fig. 4. The Dutch Auction Protocol

Multiple bids – This rule obliges the auctioneer to inform the buyers, whenever a collision comes about, about the collision and to restart the bidding round at a higher price (in this case, 120% of the collision price). Notice that X will hold all the utterances at scene *dutch* and state w_4 issued by buyer agents that bid for an item It at price P at time T_0 after the last offer. We obtain the last offers by checking that there are no further offers whose time-stamps are greater than the time-stamp of the first one. If the number of illocutions in X is greater than one, the rule fires the obligation above:

$$\left(\begin{array}{c} X = \{\, \alpha_0 | \alpha_1 \wedge \neg\, (\alpha_2 \ \wedge\ \mathtt{sat}(T_2 > T_1)) \ \wedge\ \mathtt{sat}(T_0 > T_1) \,\} \wedge \\ \mathtt{sat}(|\ X\ | > 1) \end{array} \right) \rightsquigarrow \left(\begin{array}{c} \oplus \alpha_3 \bullet \oplus \alpha_4 \bullet \\ \oplus (P_2 > P * 1.2) \end{array} \right)$$

$$where \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, collision(It, P), T_2)) \\ \alpha_4 = obl(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \tag{12}$$

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

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

$$where \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)) \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)) \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, sold(It, P, A_1), T_4)) \end{cases} \tag{13}$$

Prevention – We must prevent agents from issuing bids they cannot afford, that is, their credit is insufficient. The rule below states that if agent Ag's credit is less than P (the last offer the auctioneer called for item It, at state

w_3 of scene *dutch*), then agent Ag is prohibited to bid.

$$\begin{array}{c}(\,\alpha_0 \wedge \neg\,(\alpha_1\ \wedge\ \mathtt{sat}(T2 > T))\wedge oav(Ag, credit, C)\ \wedge\ \mathtt{sat}(C < P)\,) \rightsquigarrow (\,\oplus\alpha_2\,)\\ where\ \begin{cases}\alpha_0 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T))\\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T_2))\\ \alpha_2 = prh(dutch, w_4, inform(A, buyer, Au, auct, bid(It, P_2), T_3))\end{cases}\end{array} \tag{14}$$

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

$$\begin{array}{c}\left(\begin{array}{c} X = \left\{\begin{array}{l|l}\alpha_0 & \alpha_1 \wedge \mathtt{sat}(T_0 > T_1)\wedge \\ & \neg\,(\alpha_2\ \wedge\ \mathtt{sat}(T_2 > T_1))\end{array}\right\}\wedge \\ oav(A_1, credit, C)\wedge \\ \mathtt{sat}(|\ X\ | = 1)\wedge \mathtt{sat}(C < P)\end{array}\right) \rightsquigarrow \left(\begin{array}{c}\ominus oav(A_1, credit, C)\bullet \\ \oplus oav(A_1, credit, C - P * 0.1)\bullet \\ \oplus\alpha_3\end{array}\right)\\ where\ \begin{cases}\alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0))\\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)),\\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2))\\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, offer(It, P * 1.2), T_3))\end{cases}\end{array} \tag{15}$$

No bids/New Price – We must check if there were no bids and if the next price is greater than the reservation price. If so, we must add an obligation for the auctioneer to start a new bidding round. Rule 16 checks that the current scene state is w_5, the last offer occurred before w_5 and whether the new price is greater than reservation price. If so, the rule adds the obligation for the auctioneer to offer the item at a lower price. By retrieving the last offer we gather the last offer price. By checking the *oav* predicates we gather the values of the reservation price and the decrement rate for item *It*.

$$\begin{array}{c}\left(\begin{array}{c} ctr(dutch, w_5, T_n)\wedge \alpha_0\wedge \\ \neg\,(\alpha_1\ \wedge\ \mathtt{sat}(T_2 > T))\wedge \mathtt{sat}(T_n > T)\wedge \\ oav(IT, reservation\_price, RP)\wedge \\ oav(IT, decrement\_rate, DR)\ \wedge \\ \mathtt{sat}(RP < P - DR)\end{array}\right) \rightsquigarrow (\,\oplus\alpha_2 \bullet \oplus(P2 = P - DR)\,)\\ where\ \begin{cases}\alpha_0 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T))\\ \alpha_1 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T_2))\\ \alpha_2 = obl(dutch, w5, inform(Au, auct, all, buyer, offer(IT, P2), T_3))\end{cases}\end{array} \tag{16}$$

No bids/withdrawal – We must check if there were no bids and the next price is less than the reservation price; if so we add the obligation for the auctioneer to withdraw the item. Rule 17 checks that the current institutional state is w_5, the last offer occurred before w_5 and whether the new offer price is greater than reservation price. If the LHS holds, the rule fires to add the obligation for the auctioneer to withdraw the item. By checking the last offer we gather the last offer price. By checking the *oav* predicates we gather the values of the reservation price and the decrement rate for the price of item

It:

$$\left(\begin{array}{c} ctr(dutch, w_5, T_n) \wedge \alpha_0 \wedge \\ \neg\,(\alpha_1 \ \wedge\ \mathtt{sat}(T_2 > T)) \wedge \mathtt{sat}(T_n > T) \wedge \\ oav(It, reservation\_price, RP) \wedge \\ oav(It, decrement\_rate, DR) \ \wedge\ \mathtt{sat}(RP \geq P - DR) \end{array}\right) \rightsquigarrow (\oplus \alpha_2) \tag{17}$$

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

This example illustrates how our language addresses the pragmatic concerns raised in section 2.

6 Expressiveness Analysis

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

6.1 Conditional Deontic Logic with Deadlines

As shown in the BNF definition of figure 5, a norm as defined in [48] is composed by several parts. The norm condition is the declaration of the context in which

```
                    NORM ::= NORM_CONDITION
                             VIOLATION_CONDITION
                             DETECTION_MECHANISM
                             SANCTIONS
                             REPAIRS
VIOLATION_CONDITION ::= formula
DETECTION_MECHANISM ::= {action expressions}
            SANCTIONS ::= PLAN
               REPAIRS ::= PLAN
                  PLAN ::= action expression | action expression ; PLAN
```

Fig. 5. BNF of Norms from [48]

the norm applies. The other fields in the norm description are; 1) the *violation condition* which is a formula defining when the norm is violated, 2) the *detection mechanism* which describes the mechanisms included in the agent platform that can be used for detecting violations, 3) the *sanctions* which defines the actions that are used to punish the agent(s) violation of the norm, and 4) the *repairs* which is a set of actions that are used for recovering the system after the occurrence of a violation. As the definition of figure 6 shows, norms can be deontic notions as either permissions, obligations or prohibitions. Furthermore, norms can be related to actions or to predicates (states). The former restrict or allow

$$
\begin{aligned}
\text{NORM\_CONDITION} &::= N(S\ \langle \mathsf{IF}\ C\rangle) \mid \mathsf{OBLIGED}(a\ \mathsf{ENFORCE}(N(a, S\langle \mathsf{IF}\ C\rangle))) \\
N &::= \mathsf{OBLIGED} \mid \mathsf{PERMITTED} \mid \mathsf{FORBIDDEN} \\
J &::= (a, P) \mid (a\ \mathsf{DO}\ A) \\
S &::= J \mid J\ \mathsf{TIME} \mid J\ \mathsf{ACTION} \\
C &::= formula \\
P &::= predicate \\
A &::= action\ expression \\
\mathsf{TIME} &::= \mathsf{BEFORE}\ D \mid \mathsf{AFTER}\ D \\
\mathsf{ACTION} &::= \mathsf{BEFORE}\ J \mid \mathsf{AFTER}\ J
\end{aligned}
$$

Fig. 6. BNF of Norm Conditions

the actions that a set of agents can perform, the latter constrain the results of the actions that a set of agents can perform. This results are predicates that can be done true or false. It is forbidden that tom performs the action of smoke (FORBIDDEN (tom DO $smoke$)) and it is forbidden that tom brings about that the air is polluted (FORBIDDEN ($tom, polluted(air)$)) are two examples of the types of norm stated above.

Through the condition (C) and temporal operators (BEFORE and AFTER), norms can be made only applicable to certain situations. Conditions and temporal operators are considered optional. Temporal operators can be applied to a deadline (D) or to an action or predicate (J).

We now explain the translation of the norms presented above into our rule language. Since we consider illocutions as the only actions that can be performed in a electronic institution, actions need to be translated into illocutions uttering that the action has been done.We call *contextualisation* to this process. Table 1 shows the translation of general norms into our rules. The permission of an

Norms from [48]	Institutional rules
PERMITTED$((A\ \mathsf{DO}\ utter(S, W, I)))$	$per(S, W, I)$
FORBIDDEN$((A\ \mathsf{DO}\ utter(S, W, I)))$	$prh(S, W, I)$
OBLIGED$((A\ \mathsf{DO}\ utter(S, W, I)))$	$obl(S, W, I)$

Table 1. Translation of general norms into predicates

action can be translated into a rule the converts the attempt to utter the I illocution at state W of scene S ($att(S, W, I)$) into the result of the illocution being uttered ($utt(S, W, I)$). The prohibition of an action can be translated into a rule that ignores the attempt to utter the illocution, and optionally can sanction the violation ($vio(id)$). For space reasons, we use the predicate vio to represent that the proper sanctions and repairs are executed. We assume that there exists another rule with the vio predicate on the LHS and sanctions and repairs on the RHS. The obligation of an action needs to be translated into two rules. The former rule sanctions the obliged agents if they do not utter the

expected illocution in the scene and state in which they are obliged to do it. The latter rule removes the obligation once the obliged action has been done.

Table 2 shows the translation of conditional norms into our rules. This trans-

Norms from [48]	Institutional rules
PERMITTED$((A$ DO $utter(S, W, I))$ IF $C)$	$C \rightsquigarrow per(S, W, I)$
FORBIDDEN$((A$ DO $utter(S, W, I))$ IF $C)$	$C \rightsquigarrow prh(S, W, I)$
OBLIGED$((A$ DO $utter(S, W, I))$ IF $C)$	$C \rightsquigarrow obl(S, W, I)$

Table 2. Translation of conditional norms into rules

lation can be done in a similar way to the translation done in the previous table but adding a condition (C) on the LHS of the rule.

Table 3 shows the translation of norms with the BEFORE *time* construct into

Norms from [48]	Institutional rules
PERMITTED$((A$ DO $utter(S, W, I))$ BEFORE $D)$	$per(S, W, I) \wedge \texttt{sat}(T < D)$
FORBIDDEN$((A$ DO $utter(S, W, I))$ BEFORE $D)$	$prh(S, W, I) \wedge \texttt{sat}(T < D)$
OBLIGED$((A$ DO $utter(S, W, I))$ BEFORE $D)$	$obl(S, W, I) \wedge \texttt{sat}(T < D)$

Table 3. Translation of "BEFORE *time*" norms into rules

our rules. This translation can be done likewise the translation done in the table 1 but adding in the LHS of the rule the condition that the time in which the attempt is done (T) has to be less that the deadline (D). Now in the translation of obligations we need three rules: one to sanction the agents that do not utter the expected illocution before the deadline, other to sanction the agents that utter the expected illocution late and another rule to remove the obligation if the illocution is uttered before the deadline.

Table 4 shows the translation of norms with the construct AFTER *time* into our rules. This translation can be done in a similar way to the translation done

Norms from [48]	Institutional rules
PERMITTED$((A$ DO $utter(S, W, I))$ AFTER $D)$	$per(S, W, I) \wedge \texttt{sat}(T > D)$
FORBIDDEN$((A$ DO $utter(S, W, I))$ AFTER $D)$	$prh(S, W, I) \wedge \texttt{sat}(T > D)$
OBLIGED$((A$ DO $utter(S, W, I))$ AFTER $D)$	$obl(S, W, I) \wedge \texttt{sat}(T > D)$

Table 4. Translation of "AFTER *time*" norms into rules

in the table 1 but adding to the LHS of the rule the condition that the time in which the attempt is done (T) has to be greater that the deadline (D). Now in the translation of obligations we only need one rule to remove the obligation if the illocution is uttered after the specified time. In the current implementation

of electronic institutions obligations must be satisfied the first time the agents are in the expected scene and state. However, as we do not assume that, this norm cannot be sanctioned.

Table 5 shows the translation of norms with the construct BEFORE *action* into our rules. The translation of a permission before another utterance is done

Norms from [48]	Institutional rules
PERMITTED((A DO $utter(S, W, I)$) BEFORE (B DO $utter(S_2, W_2, I_2)$))	$per(S, W, I) \wedge utt(S_2, W_2, I_2) \rightsquigarrow \ominus per(S, W, I)$
FORBIDDEN((A DO $utter(S, W, I)$) BEFORE (B DO $utter(S_2, W_2, I_2)$))	$per(S, W, I) \wedge utt(S_2, W_2, I_2) \rightsquigarrow \ominus prh(S, W, I)$
OBLIGED((A DO $utter(S, W, I)$) BEFORE (B DO $utter(S_2, W_2, I_2)$))	$per(S, W, I) \wedge utt(S_2, W_2, I_2) \rightsquigarrow \ominus obl(S, W, I)$

Table 5. Translation of "BEFORE *action*" norms into rules

by adding two rules that transform an attempt to utter into the illocution being uttered, first if the second action has not been performed yet or if the second illocution has been uttered but after the permitted illocution. The translation of a prohibition before another utterance is also done by adding two rules similar to the rules for the permitted before case but changing the RHS to sanction the violation. The translation of an obligation before another utterance is done by means of four rules; a) sanctions when the permitted illocution has not been uttered and the deadline illocution has been uttered; b) sanctions when the deadline illocution has been uttered before the permitted illocution; c) removes the obligation if it is fulfilled and the deadline illocution has not been uttered; and d) removes the obligation if it is fulfilled before the deadline illocution has been uttered.

Table 6 shows the translation of norms with the AFTER *action* construct into

Norms from [48]	Institutional rules
PERMITTED((A DO $utter(S, W, I)$) AFTER (B DO $utter(S_2, W_2, I_2)$))	$utt(S_2, W_2, I_2) \rightsquigarrow \oplus per(S, W, I)$
FORBIDDEN((A DO $utter(S, W, I)$) AFTER (B DO $utter(S_2, W_2, I_2)$))	$utt(S_2, W_2, I_2) \rightsquigarrow \oplus prh(S, W, I)$
OBLIGED((A DO $utter(S, W, I)$) AFTER (B DO $utter(S_2, W_2, I_2)$))	$utt(S_2, W_2, I_2) \rightsquigarrow \oplus obl(S, W, I)$

Table 6. Translation of "AFTER *action*" norms into rules

our rules. The translation of these type of norms is simply done by checking if the time when the attempt to utter an illocution is greater than the time of the deadline illocution.

Hence, the norms defined in [48] can be translated into normative rules by adding the violation condition into the LHS of the rule and sanctions and repairs

into the RHS as the following rule schema shows:

$$VC \rightsquigarrow S \bullet R$$

where VC is the violation condition, S and R stands respectively for sanctions and repairs, all of them extracted from the norm.

6.2 Z Specification of Norms

Although the work in [32,30] proposes a framework that covers several topics of normative multi-agent systems we shall focus on its definition of norm. Figure 7 shows a norm from [30] composed of several parts. In the schema,

Norm

$addresses, beneficiaries : \mathbb{P}\, Agent$
$normativegoals, rewards, punishments : \mathbb{P}\, Goal$
$context, exceptions : \mathbb{P}\, EnvState$

$normativegoals \neq \emptyset;\ addresses \neq \emptyset;\ context \neq \emptyset$
$context \cap exceptions = \emptyset;\ rewards \cap punishments = \emptyset$

Fig. 7. Z Definition of a Norm from [30]

addressees stands for the set of agents that have to comply with the norm; *beneficiaries* stands for the set of agents that profit from the compliance of the norm; *normativegoals* stands for the set of goals that ought to be achieved by addressee agents; *rewards* are received by addressee agents if they satisfy the normative goals; *punishments* are imposed to addressee agent when they do not satisfy the normative goals; *context* specifies the preconditions to apply the norm and *exceptions* when it is not applicable. Notice that a norm must always have addressees, normative goals and a context; *rewards* and *punishments* are disjoint sets, and *context* and *exceptions* too.

A norm from [30] can be translated into the following rule schema to detect its violation:

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

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

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

where *rewards* are also contextualised actions obtained from the norm. This rule specifies that a reward should be given when
addressee agents comply with the norm, which is when the norm is applicable and the contextualised normative goal ($goal'$) has been achieved.

6.3 Event Calculus

Event calculus is used in [4] for the specification of protocols. Event calculus is a formalism to represent reasoning about actions or events and their effects in a logic programming framework and is based on a many-sorted first-order

Predicate	**Meaning**
happens(Act, T)	Action Act occurs at time T
initially($F = V$)	The value of fluent F is V at time 0
holdsAt($F = V$, T)	The value of fluent F is V at time T
initiates(Act, $F = V$, T)	The occurrence of action Act at time T initiates a period of time for which the value of fluent F is V
terminates(Act, $F = V$, T)	The occurrence of action Act at time T terminates a period of time for which the value of fluent F is V

Fig. 8. Main Predicates of Event Calculus

predicate calculus. Predicates that change with time are called *fluents*. Figure 9 shows how obligations, permissions, empowerments, capabilities and sanctions are formalised by means of fluents – prohibitions are not formalised in [4] as a fluent since they assume that every action not permitted is forbidden by default.

An example of obligation specified in event calculus extracted from [4] is shown in figure 10. The obligation that C revokes the floor holds at time T if C enacts the role of chair and the floor is granted to someone else different from the best candidate.

If we translate all the *holdsAt* predicates into *uttered* predicates, we can translate the obligations and permissions of the example by including the rest of conditions in the LHS of the normative rules. However, since there is no concrete definition of norm, we cannot state that the approach in [4] is fully translatable into our rules.

Although event calculus models time, the deontic fluents specified in the example of [4] 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.

Fluent	Domain	Meaning
$requested(S, T)$	boolean	subject S requested the floor at time T
$status$	$\{free, granted(S, T)\}$	the status of the floor: $status = free$ denotes that the floor is free whereas $status = granted(S, T)$ denotes that the floor is granted to subject S until time T
$best\_candidate$	agent identifiers	the best candidate for the floor
$can(Ag, Act)$	boolean	agent Ag is capable of performing Act
$pow(Ag, Act)$	boolean	agent Ag is empowered to perform Act
$per(Ag, Act)$	boolean	agent Ag is permitted to perform Act
$obl(Ag, Act)$	boolean	agent Ag is obliged to perform Act
$sanction(Ag)$	$\mathbb{Z}^*$	the sanctions of agent Ag

Fig. 9. Main Fluents from [4]

$$\begin{aligned}&\mathsf{holdsAt}(obl(C, revoke\_floor(C)) = \mathsf{true}, T) \leftarrow \\ &role\_of(C, chair), \mathsf{holdsAt}(status = granted(S, T'), T), (T \geq T'), \\ &\mathsf{holdsAt}(best\_candidate = S', T), (S \neq S')\end{aligned}$$

Fig. 10. Example of Obligation in Event Calculus

6.4 Hybrid Metric Interval Temporal Logic

In [9] we find a proposal to represent norms via rules written in a modal logic with temporal operators called hyMITL$^{\pm}$. It combines CTL$^{\pm}$ with Metric Interval Temporal Logic (MITL) as well as features of hybrid logics. That proposal uses the technique of formula progression from the TLPlan planning system to monitor social expectations until they are fulfilled or violated.

Formula 18 below shows an example of rule in hyMITL. This rule states that if the current state is such that consumer c has just made a payment for a service, and the current state is within one week after the time the payment is made (time t) then weekly reports will be sent during the next 52 weeks until provider p optionally cancels the order.

$$\begin{aligned}&\mathsf{AG}^{+}(Done(c, make\_payment(c, p, amount, prod\_num)) \wedge [t, t + 1\text{week}) \rightarrow \\ &\downarrow^{week} w.(\downarrow^{week} cw.(\neg\, \mathsf{F}^{-}_{[-0,cw]} Done(p, send\_report(c, prod\_num, cw)) \rightarrow \\ &\qquad \mathsf{F}^{+}_{[+0,cw+1\text{week}]} Done(p, send\_report(c, prod\_num, w))) \\ &\qquad \mathsf{W}^{+}_{[w+1\text{week}, w+53\text{weeks}]} \\ &\qquad Done(c, cancel\_order(c, p, prod\_num))))\end{aligned} \tag{18}$$

Rule 19 shows the translation of the previous hyMITL$^{\pm}$ rule into our language. We calculate the number of weeks since the last utterance of payment was made and the time in which this week ends. If the number of weeks is less than 52 and the report for that week has not been sent then the agent being paid is obliged to send a report before the end of the week.

$$\left(\begin{array}{c} \alpha_0 \wedge \neg\,(\alpha_1 \ \wedge\ \mathtt{sat}(T_1 > T_0)) \wedge \\ \mathit{current\_date}(T_n) \wedge \\ \mathtt{sat}(W = \mathit{trunc}((T_n - T_0)/(6048 * 10^5))) \wedge \\ \mathtt{sat}(W < 52) \wedge \neg\, \alpha_2 \wedge \\ \mathtt{sat}(T_{end\_w} = T_0 + (W + 1) * (6048 * 10^5)) \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \oplus\alpha_3 \bullet \\ \oplus(T_i < T_{end\_w}) \end{array}\right) \tag{19}$$

$$\mathit{where} \begin{cases} \alpha_0 = \mathit{utt}(\mathit{paymt}, w_0, \mathit{inform}(C, \mathit{cust}, P, \mathit{payee}, \mathit{pay}(Am, \mathit{Prod}), T_0)) \\ \alpha_1 = \mathit{utt}(\mathit{report}, w_1, \mathit{inform}(C, \mathit{cust}, P, \mathit{payee}, \mathit{cancel}(\mathit{Prod}), T_1)) \\ \alpha_2 = \mathit{utt}(\mathit{report}, w_2, \mathit{inform}(P, \mathit{payee}, C, \mathit{cust}, \mathit{snd\_rep}(R, W), T_2)) \\ \alpha_3 = \mathit{obl}(\mathit{report}, w_2, \mathit{inform}(P, \mathit{payee}, C, \mathit{cust}, \mathit{snd\_rep}(R, W), T_3)) \end{cases}$$

Our rules are equivalent to $\mathsf{AG}^+(LHS \rightarrow \mathsf{X}^+ RHS)$ where LHS and RHS are atomic formulae without temporal operators. As we build the next state of affairs by applying the operations on the RHS of the fired rules, we cannot use any other temporal operator in the RHS of our rules. Furthermore, since our state of affairs has non-monotonic features we cannot reason over the past of any formulae. We can only do it with predicates with time-stamps, like the *utt* predicate, that are not removed from the state of affairs.

We can capture the meaning of the X^- operator when it is used on the LHS of the hyMITL rule: $\mathsf{X}^-\phi$ is intuitively equivalent to $\mathit{ctr}(S, W, T_s) \wedge \phi(T_0) \wedge \mathtt{sat}(T_0 = T_s - 1)$. Moreover, we can also translate the U^+ operator when it is used in the RHS of the hyMITL rule: $\phi\ \mathsf{U}^+\psi$ is roughly equivalent to $\psi \rightsquigarrow \ominus\phi$. Although we cannot use all the temporal operators on the RHS of our rules, we can obtain equivalent results by imposing certain restrictions in the set of rules. $\mathsf{F}^+\phi$ can be achieved if $\oplus\phi$ appears on the RHS of a rule and it is possible that the rule fires. $\mathsf{G}^+\phi$ can be achieved after ϕ is added and no rule that could fire removes it. Time intervals can be translated into comparisons of time-points as shown in the previous example.

6.5 Social Integrity Constraints

In [1] the language Social Integrity Constraints (SIC) is proposed. This language's constructs check whether some events have occurred and some conditions hold to add new expectations, optionally with constraints. An example of a SIC construct is:

$$\left(\begin{array}{l} \mathbf{H}(\mathit{request}(B, A, P, D, T_r)) \wedge \\ \mathbf{H}(\mathit{accept}(B, A, P, D, T_a)) \wedge \\ T_r < T_a \end{array}\right) \rightarrow \mathbf{E}(\mathit{do}(A, B, P, D, T_d)) : T_d < T_a + \tau$$

The construct above intuitively means "if agent B sent a request P to agent A at time T_r in the context of dialogue D, and A sent an *accept* to B's request at a later time T_a, then A is expected to do P before a deadline $T_a + \tau$". The translation of SICs is based on translating events ($\mathbf{H}$) into our *att* predicates. Since we also allow predicates to be restricted by constraints, expectations can

be translated directly into obligations as the next rule shows:

$$\left(\begin{array}{c} utt(D,\, W_0, request(B,\, R,\, A,\, R',\, P,\, T_r))\wedge \\ utt(D,\, W_1, accept(A,\, R',\, B,\, R,\, P,\, T_a))\wedge \\ \mathtt{sat}(T_r < T_a) \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \oplus obl(D,\, W_2, inform(A,\, R',\, B,\, R,\, P,\, T_d))\bullet \\ \oplus(T_d < T_a + \tau) \end{array}\right) \tag{20}$$

Although syntactically their language is very similar to ours, they are semantically different. Different from their use of abduction and Constraint Handling Rules (CHR) to execute their expectations, we use a forward chaining approach. Despite the fact that expectations they use are quite similar to obligations, SIC lacks further deontic notions such as permissions or prohibitions. Furthermore, although they mention how expectations are treated, that is, what happens when an expectation is fulfilled or when it is not, and state the possibility of SICs being violated, no mechanism to regulate agents' behaviour like the punishment of offending agents or repairing actions are offered.

6.6 Object Constraint Language

The work in [15] proposes the Object Constraint Language (OCL) for the specification of artificial institutions. The expression below shows an example of norm written in OCL:

$$\begin{array}{l} \mathsf{within}\ h : AuctionHouse \\ \mathsf{on}\ e : InstitutionalRelationChange(h.dutchAuction, \\ \qquad auctioneer, created) \\ \mathsf{if}\ true\ \mathsf{then} \\ \mathsf{foreach}\ agent\ \mathsf{in} h.employee \rightarrow \\ \qquad select(em \mid e.involved \rightarrow contains(em)) \\ \ \mathsf{do}\ \mathtt{makePendingComm}(agent, \\ \qquad DutchInstAgent(notSetCurPrice(\\ \qquad h.dutchAuction.id, \\ \qquad ?p[?p < h.agreement.reservationPrice]), \\ \qquad < now, now + time\_of(e1 : InstStateChange(\\ \qquad h.dutchAuction, OpenDA, ClosedDA)) >, \forall)) \end{array} \tag{21}$$

This norm commits the auctioneer not to declare a price lower than the agreed reservation price. As shown in section 5.4, we can also express (rule 17) the case that the auctioneer is obliged to withdraw the good when the call price becomes lower than the reservation price. As for [15], we cannot perform an exhaustive analysis of the language because neither the syntax nor the semantics are specified.

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

7 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, contributions from sociology highlight the importance of norms in agent behaviour (*e.g.*, [8,7,45]) or analyse the emergence of norms in multi-agent systems (*e.g.*, [50,42]). On the other hand, logic-oriented contributions focus on the deontic logics required to model normative modalities along with their paradoxes (*e.g.*, [49,2,28]). 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 (*e.g.*, [10,6,48,14]).

Vázquez-Salceda *et al.* [48] propose the use of a deontic logic with deadline operators. These operators specify the time or the event after (or before) which a norm is valid. This deontic logic includes obligations, permissions and prohibitions, possibly conditional, over agents' actions or predicates. In their model, 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 [48]: one is that we provide an implementation for our rules and the other is that we offer a more expressive language with constraints over norms (*e.g.*, an agent can be obliged to pay an amount greater than some fixed value).

Fornara *et al.* [14] propose the use of norms partially written in OCL, the Object Constraint Language which is part of UML (Unified Modelling Language) [37]. 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 that new or unexpected, risky actions to be carried out. Their within, on and if clauses can be encoded into the 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 in the RHS of our rules since they are the actions to be applied to a set of agents.

López y López *et al.* [31] 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 the rules of Fornara *et al.* [14]. 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.* [4] propose the use of event calculus for the specification of protocols. Obligations, permissions, empowerments, capabilities and sanctions are formalised by means of fluents – these are predicates that change with time. Prohibitions are not formalised in [4] 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. In [43] (previous to the work of Artikis *et al.* [4]), Stratulat *et al.* also used event calculus to model obligations, permissions, prohibitions and violations. Similar to the work of Artikis *et al.*, that proposal lacks an representation of time that would be easily processed by agents.

Michael *et al.* [34] 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.

Kollingbaum [24] proposes a language for the specification of normative concepts (*i.e.*, obligations, prohibitions and permissions) and a programming language for norm-governed reasoning agents. The normative concepts and the programming language are given their operational semantics via the NoA Agent Architecture [25,26] using Java to explain the meaning of each construct. This approach addresses practical reasoning agents developed using their language and architecture – although the approach is practical and has clear advantages such as the possibility to check for norm conflicts and consistency, heterogeneous agents cannot be accommodated. Furthermore, there is no indication of how the proposal adapts to a distributed scenario, as only individual agents are addressed.

In [1] the language Social Integrity Constraints (SIC) is proposed. This language's constructs check whether some events have occurred and some conditions hold to add new expectations, optionally with constraints. Although syntactically their language is very similar to ours, they are semantically different. Different from their use of abduction and Constraint Handling Rules (CHR) to execute their expectations, we use a forward chaining approach. Despite the fact that expectations they use are quite similar to obligations and they mention how expectations are treated, that is, what happens when an expectation is fulfilled or when it is not, and state the possibility of SICs being violated, no mechanism to regulate agents' behaviour like the punishment of offending agents or repairing actions are offered.

The work in [16] reports on the translation of the normative language presented in [48] into Jess rules to monitor and enforce norms. This language captures the deontic notions of permission, prohibition and obligation in several cases: absolute norms, conditional norms, norms with deadline and norms in temporal relation with another event. Absolute norms are directly translated into Jess facts; conditional norms are directly translated into rules that add the deontic facts when the condition holds; norms with deadline are translated into rules that add conditional norms after the deadline has passed. Finally, norms in

temporal relation with other events are translated into rules that check if those events have occurred.

Our proposal bears strong similarities with the work reported in [29] where norms are represented as rules of a production system. We notice that our rules can express their notions of contracts and their monitoring (*i.e.*, fulfilment and violation of obligations). However, in [29] constraints can only be used to depict the right-hand side of a rule, that is, the situation(s) when a rule is applicable – constraints are not manipulated the way we do. Furthermore, in that work there is no indication as to how individual agents will know about their normative situation; a diagram introduces the architecture, but it is not clear who/what will apply the rules to update the normative aspects of the system nor how agents synchronise their activities.

8 Conclusions, Discussion and Future Work

In this paper we have introduced a formalism for the explicit management of the normative positions of agents in electronic institutions. Electronic institutions define a computational model that mediates and regulates the interaction of a community of agents. The classical model of electronic institution proposed in [12] is strict in the sense that only permitted illocutions are accepted in the interactions. We propose a language to implement and extend the notion of electronic institution by providing them with several flavours of deontic notions.

Ours is a rule language in which constraints can be specified and changed at run-time, conferring expressiveness and precision on our constructs. The semantics of our formalism defines a production system in which rules are exhaustively applied to a state of affairs, leading to the next state of affairs. The normative positions are updated via rules, depending on the messages agents send.

Our formalism addresses the points of a desiderata for normative languages introduced in section 2: We explicitly manage normative positions with our language as facts of our production system. We have explored the pragmatics and generality of our proposal in sections 3.5 and 5.4 by introducing the type of expressions that can be specified with the language and by specifying a version of the Dutch Auction protocol. We also illustrate how our language can provide other (higher-level) normative languages with a computational model (*i.e.*, an *implementation*) thus making it possible for some normative languages proposed with more theoretical concerns in mind to become executable. Our language is rule-based thus addressing the requirement that norm-oriented languages should be declarative laid out in section 2.

We propose norm-oriented programming as a paradigm to regulate the interactions among the components of a system. We notice that it is complementary to other paradigms that focus on regulating the internal processes of these components such as agent-oriented programming [41]. We intend to tackle the engineering of regulation mechanisms of open MAS from a social perspective.

The main advantage of using our language, instead of standard production systems, to specify and monitor the normative position of the agents conforming

a MAS is the inclusion of constraint solving techniques in the semantics to handle with constrained predicates.

As for future work, rather than just considering events as utterances of illocutions (some of them reporting on actions, such as the "bid" message in the example of section 5.4), we would like to generalise our language to cope with arbitrary actions, as this would allow us to address any multi-agent system. We would also like to extend the syntax and semantics of our language to support temporal operators for the management of time.

We also want to investigate the verification of norms (along the lines of our work in [46]) expressed in our rule language, with a view to detecting, for instance, obligations that cannot be fulfilled, prohibitions that will prevent progress, inconsistencies and so on. We are currently investigating tools to help engineers preparing their rules – these are norm editors that will support the design and verification of norm-oriented electronic institutions.

Acknowledgements – The authors want to thank Pere Garcia and Pablo Noriega for their helpful comments. This work was partially funded by the Spanish Science and Technology Ministry as part of the Web-i-2 project (TIC-2003-08763-C02-00). García-Camino enjoys an I3P grant from the Spanish Council for Scientific Research (CSIC).

References

1. Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Specification and Verification of Agent Interactions using Integrity Social Constraints. Technical Report DEIS-LIA-006-03, Università degli Studi di Bologna, October 2003.
2. Carlos E. Alchourron and Eugenio Bulygin. The Expressive Conception of Norms. In R. Hilpinen, editor, *New Studies in Deontic Logics*, pages 95–124, London, 1981. D. Reidel.
3. Krzysztof R. Apt. *From Logic Programming to Prolog*. Prentice-Hall, U.K., 1997.
4. Alexander Artikis, Lloyd Kamara, Jeremy Pitt, and Marek Sergot. 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.
5. Robert Axelrod. *The complexity of cooperation: agent-based models of competition and collaboration*. Princeton studies in complexity. Princeton University, New Jersey, 1997.
6. Guido Boella and Leendert van der Torre. Permission and Obligations in Hierarchical Normative Systems. In *Procs. 8th Int'l Conf. in AI & Law (ICAIL'03)*, Edinburgh, 2003. ACM.
7. Rosaria Conte and Cristiano Castelfranchi. Norms as Mental Objects: From Normative Beliefs to Normative Goals. In *Procs. of MAAMAW'93*, Neuchatel, Switzerland, 1993.
8. Rosaria Conte and Cristiano Castelfranchi. Understanding the Functions of Norms in Social Groups through Simulation. In N. Gilbert and R. Conte, editors, *Artificial Societies. The Computer Simulation of Social Life*, pages 252–267, London, 1995. UCL Press.

9. Stephen Cranefield. A Rule Language for Modelling and Monitoring Social Expectations in Multi-Agent Systems. Technical Report 2005/01, University of Otago, February 2005.
10. Frank Dignum. Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.
11. Mark d'Inverno and Mike Luck. Engineering AgentSpeak(L): A Formal Computational Model. *Journal of Logic and Computation*, 8(3):1–27, 1998.
12. Marc Esteva. *Electronic Institutions: from specification to development*. PhD thesis, Universitat Politecnica de Catalunya, 2003. Number 19 in IIIA Monograph Series.
13. Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, U.S.A., 1990.
14. Nicoletta Fornara, Francesco Viganò, and Marco Colombetti. A Communicative Act Library in the Context of Artificial Institutions. In *2nd European Workshop on Multi-Agent Systems*, pages 223–234, Barcelona, 2004.
15. Nicoletta Fornara, Francesco Viganò, and Marco Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In *AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM)*, Utrecht, 2005.
16. Andrés García-Camino, Pablo Noriega, and Juan-Antonio Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 667–673, Utrecht, The Nederlands, July 2005.
17. Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In M. Baldoni et al., editors, *Declarative Agent Languages and Technologies III*, volume 3904 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 89–105. Springer, Berlin Heidelberg, March 2006.
18. Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *ACM SIGecom Exchanges*, 5(5):33–40, January 2006.
19. Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. Norm Oriented Programming of Electronic Institutions. In *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'06)*, May 2006.
20. Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. Norm-Oriented Programming of Electronic Institutions: A Rule-based Approach. In *Coordination, Organization, Institutions and Norms in agent systems (COIN'06) in Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS'06)*, May 2006.
21. Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
22. Joxan Jaffar, Michael J. Maher, Kim Marriott, and Peter J. Stuckey. The Semantics of Constraint Logic Programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
23. Nick R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems*, 1:7–38, 1998.
24. Martin J. Kollingbaum. *Norm-Governed Practical Reasoning Agents*. PhD thesis, Department of Computing Science, University of Aberdeen, United Kingdom, January 2005.

25. Martin J. Kollingbaum and Tim J. Norman. NoA – A Normative Agent Architecture. In *Procs. 18th Int'l Joint Conference on Art. Intelligence (IJCAI)*, pages 1465–1466, Acapulco, Mexico, 2003. AAAI Press.
26. Martin J. Kollingbaum and Tim J. Norman. Norm Adoption in the NoA Agent Architecture. In *Procs. 2nd Int'l Joint Conf. on Autonomous Agents & Multi-Agent Systems (AAMAS 2003)*, Melbourne, Australia, 2003. ACM, U.S.A.
27. Bryan Kramer and John Mylopoulos. Knowledge Representation. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 743–759. John Wiley & Sons, 1992.
28. Alessio Lomuscio and Donald Nute, editors. *Proc. of the 7th Intl. Workshop on Deontic Logic in Computer Science (DEON'04)*, volume 3065 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2004.
29. Henrique Lopes Cardoso and Eugénio Oliveira. Towards an Institutional Environment using Norms for Contract Performance. In *Multi-Agent Systems and Applications IV co-located with 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005)*, volume 3690 of *LNAI*, pages 256–265. Springer-Verlag, 2005.
30. Fabiola López y López. *Social Power and Norms: Impact on agent behaviour*. PhD thesis, University of Southampton, June 2003.
31. Fabiola López y López and Michael Luck. A Model of Normative Multi-Agent Systems and Dynamic Relationships. In *Regulated Agent-Based Social Systems*, volume 2934 of *LNAI*, pages 259–280. Springer-Verlag, 2004.
32. Fabiola López y López, Michael Luck, and Mark d'Inverno. Constraining Autonomy Through Norms. In *1st Int'l Joint Conf on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 674–681, Bologna,Italy, 2002. ACM Press.
33. Kim Mariott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. M.I.T. Press, U.S.A., 1998.
34. Loizos Michael, David C. Parkes, and Avi Pfeffer. Specifying and monitoring market mechanisms using rights and obligations. In *Proc. AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC VI)*, New York, USA, July 2004.
35. Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997. Number 8 in IIIA Monograph Series.
36. Douglass C. North. *Institutions, Institutional Change and Economics Performance*. Cambridge University Press, 1990.
37. OMG. Unified Modelling Language. `http://www.uml.org`, May 2005.
38. Juan-Antonio Rodríguez-Aguilar. *On the Design and Construction of Agent-mediated Electronic Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001. Number 14 in IIIA Monograph Series.
39. John Searle. *Speech Acts, An Essay in the Philosophy of Language*. Cambridge University Press, 1969.
40. Marek Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.
41. Yoav Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.
42. Yoav Shoham and Moshe Tennenholtz. On Social Laws for Artificial Agent Societies: Off-line Design. *Artificial Intelligence*, 73(1-2):231–252, 1995.
43. Tiberiu Stratulat, Franoise Clérin-Debart, and Patrice Enjalbert. Norms and Time in Agent-based Systems. In *Procs 8th Int'l Conf on AI & Law (ICAIL'01)*, pages 178 – 185, St. Louis, Missouri, USA, 2001.
44. Swedish Institute of Computer Science. *SICStus Prolog*, 2006. `http://www.sics.se/sicstus`, viewed on 10 Feb 2006 at 18.16 GMT.

45. Raimo Tuomela and Maj Bonnevier-Tuomela. Norms and Agreement. *European Journal of Law, Philosophy and Computer Science*, 5:41–46, 1995.
46. Wamberto W. Vasconcelos. Norm Verification and Analysis of Electronic Institutions. In *Declarative Agent Languages and Technologies III*, volume 3476 of *LNAI*. Springer-Verlag, New York, USA, July 2004.
47. Wamberto W. Vasconcelos, David Robertson, Carles Sierra, Marc Esteva, Jordi Sabater, and Michael Wooldridge. Rapid Prototyping of Large Multi-Agent Systems through Logic Programming. *Annals of Mathematics and Artificial Intelligence*, 41(2–4):135–169, August 2004.
48. Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Implementing Norms in Multiagent Systems. In *Multiagent System Technologies: Second German Conference, MATES 2004*, volume 3187 of *LNAI*, pages 313–327, Erfurt, Germany, September 2004. Springer-Verlag.
49. Georg Henrik von Wright. *Norm and Action: A Logical Inquiry*. Routledge and Kegan Paul, London, 1963.
50. Andrew Walker and Michael Wooldridge. Understanding the emergence of conventions in multi-agent systems. In *Procs. Int'l Joint Conf. on Multi-Agent Systems (ICMAS)*, pages 384–389, San Francisco, USA, 1995.
51. Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, February 2002.