Auton Agent Multi-Agent Syst (2009) 18:186–217
DOI 10.1007/s10458-008-9059-4

Constraint rule-based programming of norms for electronic institutions

Andrés García-Camino · Juan A. Rodríguez-Aguilar · Carles Sierra · Wamberto Vasconcelos

Published online: 3 September 2008

Abstract Norms constitute a powerful coordination mechanism among heterogeneous agents. In this paper, we propose a rule language 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 to supplement (and implement) electronic institutions with norms. We also show how some normative aspects are given computational interpretation.

Keywords Norms · Constraints · Rules · Electronic institutions

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 [1]. 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 [2–5]. There is a wealth of socio-philosophical and logic-theoretical literature on the subject of norms (e.g., [6,7]). More recently, much attention has been paid

A. García-Camino (✉) · J. A. Rodríguez-Aguilar · C. Sierra
IIIA - Artificial Intelligence Research Institute, CSIC - Spanish Scientific Research Council,
Campus UAB, 08193 Bellaterra, Catalunya, Spain
e-mail: andres@iiia.csic.es

J. A. Rodríguez-Aguilar
e-mail: jar@iiia.csic.es

C. Sierra
e-mail: sierra@iiia.csic.es

W. Vasconcelos
Department of Computing Science, University of Aberdeen, AB24 3UE Aberdeen, UK
e-mail: wvasconcelos@acm.org

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 the design and execution of MASs (e.g., [8–12]).

Ideally, norms, once captured via some suitable formalism, should be directly executed, thus realising a computational, normative environment wherein agents interact. 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 [6] 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 [13]. 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 [14] or the 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 to manage their changes as agents interact via speech acts [15]. 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 several concurrent normative contexts such that, for instance, prohibitions cannot be violated in some of them and prohibitions over certain actions can be violated under penalties in some others.

Our language is rule-based and we achieve greater flexibility, expressiveness and precision than conventional production systems by allowing constraints [16,17] 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 institutions* (EIs) [18–20]. 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.

Although our language can be used for regulated MAS in general, in this paper we illustrate the use of the language for electronic institutions (cf. Sect. 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. By using it in EIs, we add new deontic notions to them such as explicit prohibitions (instead of implicit prohibitions, i.e. the absence of steps in the protocols).

Although we make use of concepts from electronic institutions [18], these are kept to a minimum. By using electronic institutions, we make our discussion more concrete and detailed, so as to allow the adaptation of our approach to other MAS models such as MOISE+ [21] and MOCHA [22]. We both extend electronic institutions [20] with a richer notion of norms (and how to manage them) and we also propose a programming language of wider appeal which combines rules and constraint-solving techniques.

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. In future work we would like to extend our approach to handle a wider range of normative notions such as power, right, duty, delegation, representation, entitlement, and so on. Additionally, methodological issues (i.e., how to obtain normative requirements from an application domain) albeit important, are not the focus of the paper.

The main contributions of this paper are:

1. A means to specify what an agent can, may, may not and ought to utter using normative positions and constraints;
2. An operational semantics to the above mentioned specification by means of rules and constraint solving techniques;
3. The application of this computational notion of norm to implement and enrich a model of regulated MAS like electronic institutions and, as illustrative example, its application to regulate the Dutch Auction; and
4. The comparison of our language with other contemporary ones and the provision of guidelines for the mapping of such languages into ours.

The structure of this paper is as follows. In the next section we present desirable properties of normative languages. We explain, in various sections, how our language addresses all these requirements. In Sect. 3 we describe the syntax and semantics of our normative language. Sect. 4 summarises electronic institutions and explains how we capture normative positions of participating agents. We put our language to use in Sects. 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 Sect. 5.4. In Sect. 6 we show how our language captures a sample of other contemporary approaches and in Sect. 7 we compare our approach with other related work. Finally, we draw conclusions and outline future work in Sect. 8.

2 Desiderata for norm-oriented languages

We aim at a language to support the specification of coordination mechanisms in MASs via norms. 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 [6] change. As many others (e.g., [4,8–10]), we also propose that hinder to agent autonomy by allowing only permitted actions to be performed is a restrictive limitation in the enactment of a MAS. In some settings, some not-permitted actions can raise kinds of behaviour which may result in more benefits

(perhaps even social ones) than the penalty received and which had not been anticipated by the designers.

In this section we identify and justify some desirable features we expect in our candidate languages:

1. *Explicit management of normative positions*—We require that our language explicitly captures different deontic notions along with their relationships. Ideally, these relationships should not to be hardwired into the semantics of the language, as new deontic notions can be included without changing the semantics.
2. *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 normative positions, e.g., whether different types of obligation should be revoked after their fulfillment or not. We require the language to be of *general purpose* so that it helps MAS designers to specify the widest possible range of normative systems.
3. *Expressive*—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.
4. *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. A more detailed discussion of the advantages of declarative languages can be found in [23, Sect. 1.2].
5. *Temporal relationships*—The violation of positive obligations cannot be sanctioned if a deadline for the action has not been established. In some settings, norms are not applicable after an established date. Thus, the language should deal with norm deadlines and norm activation times and capture temporal relationships between actions.
6. *Norm enforcement mechanisms*—When agents have the possibility to violate norms, it's designers' decision to use different mechanisms for agents to not misbehave. Thus, it is desirable that the language provides means to determine how enforcement mechanisms should be realised. Examples of enforcement mechanisms would be one that blocks illegal actions or one that punishes (or rewards) when a norm has been violated (or fulfilled).

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 [24–27]. 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

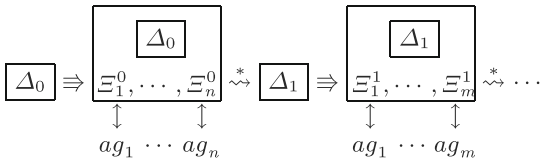

Fig. 1 Semantics as a sequence of Δ's

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.

We make use of the closed world assumption (CWA) [23] since we assume a MAS as a communication middleware that manages (and has access to) all interactions it may regulate. Therefore, we consider as false all formulae not included in the state of affairs of a MAS since we cannot regulate them.

We now introduce an example of enactment of our computational model using a Dutch Auction scenario. There are some goods that are expected to be sold to one of the agents participating in the auction. We consider these goods part of the environment. However, they are owned by one agent, enacting the role of seller, until the auctioneer, a special kind of agent that regulates the auction, finishes the process determining a winner who pays for the auctioned goods. As the state of affairs of the auction, we consider the current credit of the participants, the ownership of the goods that are part of the environment and the history of speech acts that have been considered valid at some point of the enactment of the auction. For instance, whenever the auctioneer offers a good for a given price, it has to be checked that the illocution was uttered in the correct point in the protocol by applying the rules. If this is the case, this illocution is added to the state of affairs for later checking. Continuing with the example, the participant agents may now bid for the item at the offered price. These attempts are added to the previous state of affairs conforming the current state of affairs before applying the rules to check which attempts are valid. After the application of the rules, only the valid bids remain in the state of affairs, e.g. those that the agents may afford. If it the case that there is only one bid, a winner may be determined and the obligation of the winner to pay for the goods is put in place.

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 $(ag_1, \ldots, ag_n)$. These agents can add their events $(\Xi_1^0, \ldots, \Xi_n^0)$ 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 kind of atomic formula. After an 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.

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 [23] 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 A constraint γ is a binary atomic formula $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.

We shall use $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ as a set of constraints. 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 $\alpha_i, 0 \leq i \leq n$.

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:

$$
\begin{aligned}
\mathsf{R} &::= LHS \rightsquigarrow RHS \\
LHS &::= LHS \,\&\, LHS \mid LHS \;||\; LHS \mid \texttt{not}(LHS) \mid \mathsf{Lit} \\
RHS &::= \mathsf{U}, RHS \mid \mathsf{U} \\
\mathsf{Lit} &::= \alpha \mid \texttt{sat}(\Gamma) \mid x = \{\alpha | LHS\} \\
\mathsf{U} &::= \texttt{add}(\alpha) \mid \texttt{del}(\alpha)
\end{aligned}
$$

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.

[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.

In the next section we define the semantics of each construct above, but informally, the construct α checks whether there exists an atomic formula in the state of affairs matching the atomic formula α and $\texttt{sat}(\Gamma)$ 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 $\{\alpha|LHS\}$. 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, $\{(p(A, B, C)|\texttt{sat}(B > 20)\ \texttt{\&}\ \texttt{sat}(C < 100)\}$ stands for the set of atomic formulae $p(A, B, C)$ such that B is greater than 20 and C is less than 100 and such that they satisfy the other constraints in the state of affairs. Notice that $\{p(A, B, C)|B > 20\ \&\ C < 100\}$ stands for the set of atomic formulae $p(A, B, C)$ such that B is greater than 20 and C is less than 100 without extra checking on other constraints. Notice that $\{p(A, B, C)|\texttt{constr}(B > 20)\ \texttt{\&}\ \texttt{constr}(C < 100)\}$ 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 `add`) or remove from the state of affairs (via operator `del`) atomic formulae.

3.3 Semantics of Rules

As shown in Fig. 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 [23,28]:

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. $\mathbf{s}_l(\Delta, LHS\ \&\ LHS', \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma')$ and $\mathbf{s}_l(\Delta, LHS' \cdot \sigma', \sigma'')$ hold (in this order) and $\sigma = \sigma' \cup \sigma''$.
2. $\mathbf{s}_l(\Delta, LHS\ ||\ LHS', \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma)$ or $\mathbf{s}_l(\Delta, LHS, \sigma)$ hold.
3. $\mathbf{s}_l(\Delta, \texttt{not}(LHS), \sigma)$ holds iff $\mathbf{s}_l(\Delta, LHS, \sigma)$ does not hold.
4. $\mathbf{s}_l(\Delta, \texttt{sat}(\Gamma), \sigma)$ holds iff $constrs(\Delta, \Gamma')$ and $((\Gamma' \cup \Gamma) \cdot \sigma)$ hold.

5. $\mathbf{s}_l(\Delta, x = \{\alpha | LHS\}, \sigma)$ holds iff $\sigma = \{x/\{\alpha \cdot \sigma_1, \ldots, \alpha \cdot \sigma_n\}\}$ for the largest $n \in \mathbb{N}$ such that $\mathbf{s}_l(\Delta, \alpha \,\&\, LHS, \sigma_i)$, $1 \leq i \leq n$
6. $\mathbf{s}_l(\Delta, \alpha, \sigma)$ holds iff $\alpha \cdot \sigma \in \Delta$ or $\alpha \cdot \sigma$ holds.

Cases 1 and 2 depict the semantics of atomic formulae and how their individual substitutions are combined to provide the semantics for a conjunction and a disjunction respectively. Case 3 introduces negation by failure—recall that we make use of the closed world assumption. 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, that is applied to ($\Gamma' \cup \Gamma$) will hold an assignment of variables in a Constraint Satisfaction Problem [29]. Case 5 specifies the semantics for *set constructors*: x is the set of atomic formulae that satisfy the conditions of the set constructor. Case 6 holds when an atomic formulae (a predicate or constraint) is part of the state of affairs or it is computed via the underlying programming language. This will become clearer when we discuss our implementation and give examples. It is worth noticing that, from case 1 above, the order in which conjuncts appear on the left-hand side is relevant. Our rules are means to define a deterministic program, hence the order of commands is essential.

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. $\mathbf{s}_r(\Delta, (\mathsf{U}, RHS), \Delta')$ holds iff both $\mathbf{s}_r(\Delta, \mathsf{U}, \Delta_1)$ and $\mathbf{s}_r(\Delta_1, RHS, \Delta')$ hold.
2. $\mathbf{s}_r(\Delta, \texttt{add}(\bar{\alpha}), \Delta')$ holds iff $\Delta' = \Delta \cup \{\bar{\alpha}\}$.
3. $\mathbf{s}_r(\Delta, \texttt{add}(\gamma), \Delta')$ holds iff *constrs*(Δ, Γ) and $(\Gamma \cup \{\gamma\})$ hold and $\Delta' = \Delta \cup \{\texttt{constr}(\gamma)\}$.
4. $\mathbf{s}_r(\Delta, \texttt{del}(\alpha), \Delta')$ holds iff $\Delta' = \Delta \setminus \{\alpha\}$.

Case 1 decomposes a conjunction and builds the new state by merging the partial states of each update. Case 2 caters for the insertion of atomic formulae $\bar{\alpha}$ 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' annotated with `constr`. Case 4 caters for the removal of atomic formulae (both constraints and non-constraints). We note that, from case 1 above, the order in which conjuncts appear on the right-hand side is also relevant.

To complete the definition of our 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 [30], 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 $\mathbf{s}_r$. Since classic production systems do not use of constraints and these are an important feature of our approach, our semantics can thus be seen as an extension of production systems.

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 [31] we show how a rule is interpreted in Fig. 2 as a logic program, interspersed with built-in Prolog predicates; for easy referencing, we show each clause with a number on its left.

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 1–8 and 9–12 are, respectively, adaptations of the cases depicted in Definition 10 and Definition 11.

1. $\mathbf{s}_l(\Delta, (LHS \;\&\; LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma'), \mathbf{s}_l(\Delta, LHS' \cdot \sigma', \sigma''), \sigma = \sigma' \cup \sigma''$
2. $\mathbf{s}_l(\Delta, (LHS \parallel LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS, \sigma)$
3. $\mathbf{s}_l(\Delta, (LHS \parallel LHS'), \sigma) \leftarrow \mathbf{s}_l(\Delta, LHS', \sigma)$
4. $\mathbf{s}_l(\Delta, \mathtt{not}(LHS), \sigma) \leftarrow \neg \mathbf{s}_l(\Delta, LHS, \sigma)$
5. $\mathbf{s}_l(\Delta, \mathtt{sat}(\Gamma), \sigma) \leftarrow constrs(\Delta, \Gamma'), append(\Gamma, \Gamma', \Gamma''), satisfiable(\Gamma'' \cdot \sigma)$
6. $\mathbf{s}_l(\Delta, x = \{\alpha \mid LHS\}, \{x/AllAtfs\}) \leftarrow \mathtt{findall}(\alpha \cdot \sigma, \mathbf{s}_l(\Delta, \alpha \,\&\, LHS, \sigma), AllAtfs)$
7. $\mathbf{s}_l(\Delta, \alpha, \sigma) \leftarrow \mathtt{member}(\alpha \cdot \sigma, \Delta)$
8. $\mathbf{s}_l(\Delta, \alpha, \sigma) \leftarrow \mathtt{call}(\alpha \cdot \sigma)$
9. $\mathbf{s}_r(\Delta, (\mathsf{U}, RHS), \Delta'') \leftarrow \mathbf{s}_r(\Delta, \mathsf{U}, \Delta'), \mathbf{s}_r(\Delta', RHS, \Delta'')$
10. $\mathbf{s}_r(\Delta, \mathtt{add}(\bar{\alpha}), \Delta') \leftarrow \Delta' = \{\bar{\alpha}\} \cup \Delta$
11. $\mathbf{s}_r(\Delta, \mathtt{add}(\gamma), \Delta') \leftarrow constrs(\Delta, \Gamma), satisfiable(\{\gamma\} \cup \Gamma), \Delta' = \{\mathtt{constr}(\gamma)\} \cup \Delta$
12. $\mathbf{s}_r(\Delta, \mathtt{del}(\alpha), \Delta') \leftarrow \mathtt{delete}(\Delta, \alpha, \Delta')$

Fig. 2 Interpreter for rules with constraints

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

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

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:

$$\begin{pmatrix} do(A, pay(P, B), T) \,\& \\ credit(B, X) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \mathtt{del}(do(A, pay(P, B), T)), \\ \mathtt{del}(credit(B, X)), \\ \mathtt{add}(credit(B, X + P)) \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} do(A, ext(obl(X, T2), D), T) \,\& \\ (T < D) \,\&\, \mathtt{constr}(T2 < D2) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \mathtt{del}(do(A, ext(obl(X, T2), D), T)), \\ \mathtt{del}(T2 < D2), \mathtt{add}(T2 < D) \end{pmatrix} \quad (2)$$

$$\begin{pmatrix} do(C, pay(P, B), T) \,\&\, min(D) \,\& \\ time(T) \,\&\, (P > D) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \mathtt{del}(do(C, pay(P, B))), \\ \mathtt{add}(done(C, pay(P, B), T)) \end{pmatrix} \quad (3)$$

The first example shows a rule depicting the circumstance in which it should be applied: if agent A generates the event of paying price P to agent B and the credit of the latter is X. It also shows on the *RHS* the updates to perform: we ensure the event is "consumed" (thus not triggering off the rule indefinitely) and the credit of agent B is updated to $X + P$.

The second example illustrates the management of constraints: these can be manipulated like ordinary predicates. In that example, we show that events of type *obl* (i.e. an obligation) may have associated constraints. Particularly, this rule states that if an event of extending (ext) the deadlines of all the obligations to time D occurs before the deadline D and there exists a constraint restricting the time of fulfillment of the obligations to be less than a deadline $D2$, then the event is consumed, the old constraint is removed and a constraint with the new deadline is added.

The third example illustrates how constraints can additionally be *checked* for their satisfaction: when an event of paying price P is performed by agent C and there is a formula $min(D)$ (storing a minimum price), we check that all the constraints in the state of affairs including that the amount paid is greater than the minimum ($P > D$) are satisfied. If this

is so, then we remove the event and add a record of this situation to the state of affairs. Notice that we make use of a built-in predicate $time/1$ to check the current time of the system.

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 Fig. 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 $do(Ag, Ev, T)$ 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 [24].

The language that we propose defines a standard production system enhanced with constraint satisfaction techniques in order to manage constraints as facts and to check how these constraints affects the facts they constrain. We have obtained a language to express, manage, check fulfilment and/or sanction unfulfilled normative positions, i.e. obligations, permissions and prohibitions, that are bounded with constraints. Thus, the language is useful to predict a future state of affairs with an initial state and a sequence of sets of events that occur and modify the intermediate states of affairs until we reach the final one. The limitations of the language are determined by the forward-chaining rule engine. These limitations include the inability to plan, i.e. determine the sequence of sets of events that must occur in order to reach a given state of affairs from a given initial state, or post-dicting, i.e. determine the previously unknown facts in a partial initial state given a final state and the sequence of sets of events that have occurred. However, the goal of the language is to regulate a MAS and keep track of its evolution by prediction. Post-diction and planning would be interesting for a language that an agent could use for deciding which action to perform but this is not the aim of this paper.

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 [32] to establish how interactions must be structured within an organisation. 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 which conditions. Furthermore, human institutions not only structure human interactions but also enforce individual and collective behaviour by obliging everyone to act according to the norms.

Electronic institutions (EIs) [18–20] are the electronic counterpart of human institutions—they establish the expected behaviour of agent societies. 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 [15]. An EI defines a set of constraints that articulate agent interactions, defining what speech acts are meaningful to utter.

EIs, as presented in [19] and [18], are well-understood models for MASs, with support tools[2] within which we can embed our mechanisms. EIs are a useful model to put norms in practice as this model needs a specification of what may, may not or ought to be uttered (i.e. a set of permissions, prohibitions and obligations) and it has a set of tools [33] tested in real applications [34].

In this section we introduce electronic institutions as defined in [20]. We implement them in Sect. 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 [20] 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, denoted as 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 $inform(ag_4, seller, ag_3, buyer, offer(car, 1200), 10))$. Notice that with term t we capture the temporal relationships among illocutions and we address point 5 in the desiderata of Sect. 2.

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, denoted as $\bar{\mathsf{I}}$, 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

[2] http://e-institutions.iiia.csic.es/software.html.

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 to our proposal, provided protocols can be broken down into uniquely defined states connected by edges, and 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 [20].

Although all illocutions of a protocol are *meaningful* 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.

States of affairs and states of a protocol are related concepts but should not be confused. In electronic institutions, it is possible to have many instances of protocols (or possibly, various instances of the same protocol) simultaneously enacted by agents. This means that at any one time, we could have many states of protocols represented by atomic formulae in a state of affairs.

5 Norm-oriented programming of electronic institutions

Despite successfully achieving a significant degree of openness, electronic institutions are strict in the sense that they only specify what utterances are meaningful in each moment during the enactment of interactions. As an initial step in order to enrich EIs with a wide range of normative notions, we pursue to implement rule-based electronic institutions in which deontic notions are not limited to the ones formalised in [20].

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. For instance, as we will see in Sect. 5.4, agent protocols can be specified using Finite State Machines (FSM) in order to define all the sequences of meaningful messages that the agents are able to interchange. However, we further specify with normative positions and constraints what agents may, may not or ought to utter.

In this section we use the language introduced in Sect. 3 to program electronic institutions based on the notions introduced in Sect. 4. In Subsect. 5.1 we specify how a state of affairs is represented in an EI, whereas in Subsect. 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, i.e. their normative positions. Next, we show how to implement the main normative concepts of scenes in EIs. We leave for future work how other interesting concepts such as power, right, duty, delegation, representation, entitlement, etc., can be captured.

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.

Notice that, since illocutions are uttered towards a specific other agent, normative positions over illocutions also are, i.e. an agent may be obliged to say something to another given agent.

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, 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 formula 4 to represent state change in a scene in relation to global time passing. We shall use formulae 6–8 above to represent the 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 Sect. 5.2. We show in Fig. 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 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

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

Fig. 3 Sample institutional state

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 expressiveness and generality of our language as required in Sect. 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*.

5.2.1 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 Sect. 2. We leave for future work an extensive analysis and implementation of a wide range of normative notions such as power, right, duty, delegation, representation, entitlement, etc.

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. 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:

$$att(S, W, I) \,\&\, per(S, W, I) \rightsquigarrow \mathtt{del}(att(S, W, I)), \mathtt{add}(utt(S, W, I)) \tag{4}$$

That is, permitted attempts at utterances become legal utterances.

Attempts and prohibitions can be related together by institutional rules of the form $att(S, W, I) \,\&\, prh(S, W, I) \rightsquigarrow \mathtt{del}(att(S, W, I)), sanction$ 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:

$$\left(\begin{array}{c} att(S, W, P(A_1, R_1, A_2, R_2, M, T)) \,\& \\ prh(S, W, P(A_1, R_1, A_2, R_2, M, T)) \,\& \\ oav(A_1, credit, C) \,\&\, C2 = C \times 0.9 \end{array} \right) \rightsquigarrow \left(\begin{array}{c} \mathtt{del}(att(S, W, I)), \\ \mathtt{del}(oav(A_1, credit, C)), \\ \mathtt{add}(oav(A_1, credit, C2)) \end{array} \right) \tag{5}$$

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:

$$\begin{pmatrix} att(S, W, I)\,\&\,per(S, W, I)\,\& \\ \texttt{not}(prh(S, W, I)) \end{pmatrix} \rightsquigarrow \texttt{del}(att(S, W, I))\,,\,\texttt{add}(utt(S, W, I)) \tag{6}$$

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):

$$\begin{array}{c} \begin{pmatrix} att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T))\,\& \\ (Ag_1 \neq Ag_2)\,\&\,(Ag_1 \neq Ag_3)\,\&\,(Ag_2 \neq Ag_3) \end{pmatrix} \\ \rightsquigarrow \\ \begin{pmatrix} \texttt{del}(att(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T))), \\ \texttt{add}(utt(S, W, inform(Ag_1, R, Ag_2, R', info(Ag_3, C), T))) \end{pmatrix} \end{array} \tag{7}$$

The rule above states that if an agent Ag_1, enacting role R, attempts to reveal to Ag_2, enacting role R', (private) information C about agent Ag_3, and the three variables refer to different agents, then the attempt is accepted without taking into account if it is forbidden or not. In both cases (rules 6 and 7), we can punish agents that violate prohibitions as shown in rule 5.

The semantics of obligations also depends on which rules are part of the system. These rules should be selected taking into account the semantics of the obligatory events. For instance, when an agent fulfills its obligation to pay a certain amount of money, we remove that obligation as shown in rule 8. However, an obligation to be quiet in a given situation (notice that is equivalent to a prohibition to utter anything) may not need to be consumed each time an agent is quiet and, therefore, no extra rule is required.

$$\begin{array}{c} \begin{pmatrix} att(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T))\,\& \\ per(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T))\,\& \\ \texttt{not}(prh(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T)))\,\& \\ obl(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T))\,\& \\ (Ag_1 \neq Ag_2) \end{pmatrix} \\ \rightsquigarrow \\ \begin{pmatrix} \texttt{del}(att(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T))), \\ \texttt{del}(obl(S, W, inform(Ag_1, payer, Ag_2, payee, pay(P), T))) \end{pmatrix} \end{array} \tag{8}$$

Let us consider now that the agents may be obliged to do actions before a certain deadline expressed with constraints. The designer of the MAS may choose to punish all agents who do not meet a deadline with a fee of €20. Rule 9 states that if an obligation with deadline has not been fulfilled, i.e. there exist an obligation with a constraint associated to the time, the deadline has passed, i.e. current time is greater or equal to the deadline, and we have not applied a sanction for that particular obligation, then we apply a sanction.

$$\begin{array}{c} \begin{pmatrix} obl(S, W, inform(Ag_1, R, Ag_2, R', Action, T))\,\& \\ \texttt{constr}(T < D)\,\&\,time(T2)\,\&\,(T2 \geq D)\,\& \\ \texttt{not}(sanction(obl(S, W, inform(Ag_1, R, Ag_2, R', Action, T)), (T < D))) \\ \&\,credit(Ag_1, C)\,\&\,credit(ei, C2)\,\&\,C3 = C - 20\,\&\,C4 = C2 + 20 \end{pmatrix} \\ \rightsquigarrow \\ \begin{pmatrix} \texttt{del}(credit(Ag_1, C)), \texttt{add}(credit(Ag_1, C3)), \\ \texttt{del}(credit(ei, C2)), \texttt{add}(credit(ei, C4)), \\ \texttt{add}(sanction(obl(S, W, inform(Ag_1, R, Ag_2, R', Action, T)), (T < D))) \end{pmatrix} \end{array} \tag{9}$$

These examples show that our language can be used to build norm enforcement mechanisms and address point 6 of the desiderata of Sect. 2.

5.2.2 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 an utterance:

$$att(S, W, I) \,\&\, obl(S, W, I) \,\&\, prh(S, W, I) \rightsquigarrow \texttt{add}(utt(S, W, I)) \tag{10}$$

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:

$$att(S, W, I) \,\&\, obl(S, W, I) \,\&\, prh(S, W, I) \rightsquigarrow \texttt{add}(exc(S, W, I)) \tag{11}$$

These examples illustrate how we explicitly manage normative positions of agents in our language as required in point 1 of the desiderata of Sect. 2.

5.3 Representing and enacting protocols via institutional rules

In the rest of the paper we consider scenes, presented in Sect. 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 [20] (see Sect. 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 [35]. 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 meaningful 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. Notice that although an agent may utter a meaningful illocution ($att(s, w, \mathsf{I})$ and $edge(Scene, State, IllocutionScheme, NewState)$) in a given situation, it may also need to be permitted ($per(s, w, \mathsf{I})$) and not prohibited ($\texttt{not}\ prh(s, w, \bar{\mathsf{I}})$) to do so. By "meaningful" we mean that the illocution makes sense in the context of that protocol, that is, at a particular point of the protocol, we specify via edges all possible illocutions that agents may utter at any point. Of these, some will be permitted, as explained below.

Protocols are descriptions of what may be uttered and when it can be uttered in order to have a desired meaning. When permissions are combined with attempted utterances (i.e., $att(s, w, \mathsf{I})$, as captured by formula 4 above) and approved utterances (i.e., $utt(s, w, \mathsf{I})$) 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:

$$\left(\begin{array}{c} ctr(S, W_i, T)\,\&\,att(S, W_i, I)\,\& \\ per(S, W_i, I)\,\&\,\mathtt{not}(prh(S, W_i, I))\,\& \\ edge(S, W_i, I, W_j)\,\&\,time(T2) \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \mathtt{del}(ctr(S, W_i, T)), \\ \mathtt{add}(old\_ctr(S, W_i, T)), \\ \mathtt{add}(ctr(S, W_j, T2)), \\ \mathtt{add}(utt(S, W_i, I)) \end{array}\right) \quad (12)$$

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 $ctr(S, W_j, T2)$. 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 address all the requirements introduced in Sect. 2.

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 [18]. Following [18], the fish market can be described as a place where several scenes [20] take place simultaneously, at different locations, but with some causal connection. 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, following an *open cry, sudden death, 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, and 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.

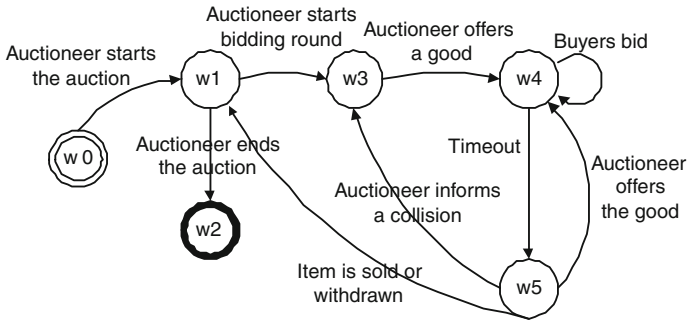

Fig. 4 The Dutch auction Protocol

5.4.1 Proposed solution

Figure 4 shows a finite state machine the protocol. Following Sect. 5.3 the protocols are represented as a set of formula of the type $edge(S, W_i, I, W_j)$ and rule 12. The situations arising in step 3 are captured in Eqs. 13–18. For formatting reasons, we will use α_i to denote atomic formulae:

Multiple bids—This rule obliges the auctioneer to inform the buyers, whenever a collision comes about, about the collision and obliges the auctioneer 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 introduces the obligation above:

$$\left(\begin{array}{c} X = \left\{\alpha_0 | \alpha_1 \,\&\, \mathtt{not}(\alpha_2 \,\&\, (T_2 > T_1)) \,\&\, (T_0 > T_1)\right\} \\ \&\, (\mathtt{size}(X) > 1) \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \mathtt{add}(\alpha_3), \mathtt{add}(\alpha_4), \\ \mathtt{add}((P_2 > P * 1.2)) \end{array}\right)$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)), \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)), \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, collision(It, P), T_2)), \\ \alpha_4 = obl(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P_2), T_3)) \end{cases} \quad (13)$$

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:

$$\left(\begin{array}{c} X = \left\{\alpha_0 | \alpha_1 \,\&\, \mathtt{not}(\alpha_2 \,\&\, (T_2 > T_1)) \,\&\, (T_0 > T_1)\right\} \& \\ (\mathtt{size}(X) = 1) \,\&\, oav(A_1, credit, C) \,\&\, (C \geq P) \end{array}\right) \rightsquigarrow (\mathtt{add}(\alpha_3))$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)), \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)), \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, sold(It, P, A_1), T_4)) \end{cases} \quad (14)$$

Prevention—We must prevent agents from issuing bids they cannot afford, that is, bids for which 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.

$$\left(\alpha_0 \,\&\, \texttt{not}(\alpha_1 \,\&\, (T2 > T)) \,\&\, oav(Ag, credit, C) \,\&\, (C < P)\right) \rightsquigarrow (\texttt{add}(\alpha_2))$$
$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T)), \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, A, buyer, offer(It, P), T_2)), \\ \alpha_2 = prh(dutch, w_4, inform(A, buyer, Au, auct, bid(It, P_2), T_3)) \end{cases} \quad (15)$$

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 its 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.

$$\left(\begin{array}{c} X = \left\{ \alpha_0 \middle| \begin{array}{l} \alpha_1 \,\&\, (T_0 > T_1) \,\& \\ \texttt{not}(\alpha_2 \,\&\, (T_2 > T_1)) \end{array} \right\} \& \\ oav(A_1, credit, C) \,\& \\ (\texttt{size}(X) = 1) \,\& \, (C < P) \,\& \\ C2 = C - P * 0.1 \end{array}\right) \rightsquigarrow \left(\begin{array}{c} \texttt{del}(oav(A_1, credit, C)), \\ \texttt{add}(oav(A_1, credit, C2)), \\ \texttt{add}(\alpha_3) \end{array}\right)$$
$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_4, inform(A_1, buyer, Au, auct, bid(It, P), T_0)), \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_1)), \\ \alpha_2 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)), \\ \alpha_3 = obl(dutch, w_5, inform(Au, auct, all, buyer, offer(It, P * 1.2), T_3)) \end{cases} \quad (16)$$

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 17 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*.

$$\left(\begin{array}{c} ctr(dutch, w_5, T_n) \,\&\, \alpha_0 \,\& \\ \texttt{not}(\alpha_1 \,\&\, (T_2 > T)) \,\&\, (T_n > T) \,\& \\ oav(IT, reservation\_price, RP) \,\& \\ oav(IT, decrement\_rate, DR) \,\& \\ (RP < (P - DR)) \end{array}\right) \rightsquigarrow \left(\texttt{add}(\alpha_2), \texttt{add}(P2 = P - DR)\right)$$
$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T)), \\ \alpha_1 = utt(dutch, w3, inform(Au, auct, all, buyer, offer(IT, P), T_2)), \\ \alpha_2 = obl(dutch, w5, inform(Au, auct, all, buyer, offer(IT, P2), T_3)) \end{cases} \quad (17)$$

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 18 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*:

$$\left(\begin{array}{c} ctr(dutch, w_5, T_n)\,\&\,\alpha_0\,\& \\ \mathtt{not}(\alpha_1\,\&\,(T_2 > T))\,\&\,(T_n > T)\,\& \\ oav(It, reservation\_price, RP)\,\& \\ oav(It, decrement\_rate, DR)\,\&\,(RP \geq (P - DR)) \end{array}\right) \rightsquigarrow \big(\mathtt{add}(\alpha_2)\big)$$

$$\text{where} \begin{cases} \alpha_0 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T)), \\ \alpha_1 = utt(dutch, w_3, inform(Au, auct, all, buyer, offer(It, P), T_2)), \\ \alpha_2 = obl(dutch, w_5, inform(Au, auct, all, buyer, withdrawn(It), T_3)) \end{cases} \quad (18)$$

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 the most common normative notions from those formalisms using our rule language. By 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. We also show that our rules can capture the patterns from rather disparate formalisms, thus fulfilling the requirement of general purpose mentioned in Sect. 2.

6.1 Conditional deontic logic with deadlines

As shown in the BNF definition of Fig. 5, a norm as defined in [36] is composed of several parts. The norm condition is the declaration of the context in which 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 define the actions that are used to punish the agent(s) violation of the norm, and (4) the *repairs* which is a set of actions used for recovering the system after the occurrence of a violation.

As the definition of Fig. 6 shows, norms can be deontic notions such as permissions, obligations or prohibitions. Furthermore, norms can be related to actions or to predicates (states). The former case restricts or allow the actions that a set of agents can perform, the latter case constrains the results of the actions that a set of agents can perform. The results of actions are represented as predicates that may hold or not. It is forbidden that Tom performs the action of smoking (FORBIDDEN (*tom* DO *smoke*)) and it is forbidden that tom brings about that

```
                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 [36]

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

Fig. 6 BNF of norm conditions

Table 1 Mapping of general norms into predicates

Norms from [36]	Rule-based construct
PERMITTED((A DO $utter(S, W, I)$))	$per(S, W, I)$
PERMITTED((A DO $utter(S, W, I)$) IF C)	$C \rightsquigarrow per(S, W, I)$
PERMITTED((A DO $utter(S, W, I)$) BEFORE D)	1. $per(S, W, I)\ \&\ \mathtt{sat}(T < D)$
	2. $\begin{pmatrix} per(S, W, I)\ \& \\ \mathtt{constr}(T < D)\ \& \\ time(T2)\ \& \\ (T2 \geq D) \end{pmatrix} \rightsquigarrow \mathtt{del}(per(S, W, I))$
PERMITTED((A DO $utter(S, W, I)$) AFTER D)	$\begin{pmatrix} time(T)\ \&\ (T > D)\ \& \\ \mathtt{not}(per(S, W, I)) \end{pmatrix} \rightsquigarrow per(S, W, I)$

the air is polluted (FORBIDDEN (tom, $polluted(air)$)) are two examples of the types of norms addressed in [36].

Through the condition (C) and temporal operators (BEFORE and AFTER), norms can be made applicable to specific situations only. Conditions and temporal operators are considered optional. Temporal operators can also be applied to a deadline (D). We refer to [37] for the formal semantics of temporal operators used in [36]. We note that states of affairs (Definition 4) loosely correspond to worlds of Kripke semantics for modal logics [38]: they both contain sets of formulae and are interrelated.

We now explain the mapping of the norms presented above into our rule language. Since we consider illocutions as the only actions that can be performed in an electronic institution, actions need to be translated into illocutions uttering that the action has been done.We call this process *contextualisation*. Table 1 shows the mapping of permissions in general norms (i.e. norms that always are active) into our rules. Prohibitions and obligations are mapped similarly. The permission for an action can be mapped into a predicate (shown in row 1 of Table 1) and added to a rule that 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 ($utt(S, W, I)$).

Row 2 of Table 1 shows the mapping of conditional norms into our rules. This mapping can be done in a similar way to the one done in the previous row but adding a condition (C) on the *LHS* of the rule. It should be pointed out that there is no one-to-one correspondence between the underlying models (i.e., semantics) of the compared approaches. However, our rules capture the same phenomenon: given C, the permission, prohibition or obligation will also hold. Importantly, only after the exhaustive application of all rules on the current state of affairs (possibly requiring a number of intermediate states) we obtain the next state of affairs, in which both C and the permission, prohibition or obligation will hold. This amounts to the logical inferences that take place in [36].

Row 3 shows the mapping of norms with the BEFORE *time* construct into our rules. This mapping can be done with two rules: one for checking if the normative position holds and if it satisfies the temporal constraints. This is similar to the mapping done in row 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); the other rule is for removing the permission after the deadline has passed. We check that the normative position has a temporal constraint and if it is not satisfied (i.e.,s the current time is greater than or equal to the deadline), we remove the normative position. In the mapping of obligations, however, we need three rules: one to sanction the agents that do not utter the expected illocution before the deadline, one 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.

Row 4 shows the mapping of permissions with the construct AFTER *time* into our rules. This can be done in a similar way to the mapping done in row 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). Notice that in the mapping 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.

In summary, the norms defined in [36] can be translated into institutional 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:

$$VC \rightsquigarrow S, R$$

where *VC* is the violation condition, *S* and *R* stands respectively for sanctions and repairs, all of them extracted from the norm. Notice that the norms defined in [36] are only applicable to a specific agent. Contrastingly, norms implemented with our rules, depending on which terms are variables, may refer to either a specific agent or all those agents enacting a role or all those agents in a scene or all those agents in any of the scenes.

6.2 Z specification of norms

Although the work depicted in [4,39] 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 [4] composed of several parts. In the schema, $addressees$ stands for the set of agents that have to comply with the norm; $beneficiaries$ stands for the set of agents that benefit from the compliance of the norm; $normativegoals$ stands for the set of goals that ought to be achieved by the 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

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 [4]

exceptions specify when the norm is not applicable. We 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 [4] can be translated into the following rule schema to detect its violation:

$$(context \ \& \texttt{not}(exception) \,\&\, \texttt{not}(goal')) \rightsquigarrow punishments$$

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:

$$(context \ \& \texttt{not}(exception) \,\&\, goal') \rightsquigarrow rewards$$

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 [8] for the specification of protocols. Event calculus [40] 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 calculus. Figure 8 shows the main predicates of Event 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 [8] 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 [8] is shown in Fig. 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 the *holdsAt* predicates into *uttered* predicates, we can translate the obligations and permissions of the example by including the remaining conditions in the *LHS* of the institutional rules. However, since there is no explicit semantics of norms in [8], we cannot state that the approach in [8] is fully translatable into our rules. But we acknowledge that the predicates in our language would need to be extended in order to capture the notion of power.

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

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 [8]

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

Fig. 10 Example of obligation in event calculus

As mentioned before, in this paper we present a rule-based language with constraint-solving capabilities and show how to regulate a MAS with some deontic notions.

Although event calculus models time, the deontic fluents specified in the example of [8] 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.

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 19 below shows an example of rule in hyMITL$^{\pm}$. 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:

$$\begin{aligned}&\mathsf{AG}^{+}(Done(c, make\_payment(c, p, amount, prod\_num)) \,\&\, [t, t + 1\text{week}) \rightarrow \\ &\downarrow^{week} w.(\downarrow^{week} cw.(\neg \mathsf{F}^{-}_{[-0,cw]} Done(p, send\_report(c, prod\_num, cw)) \rightarrow \\ &\qquad \mathsf{F}^{+}_{[+0,cw+1\text{week}]} Done(p, send\_report(c, prod\_num, w))) \\ &\qquad \mathsf{W}^{+}_{[w+1\text{week}, w+53\text{weeks}]} \\ &\qquad Done(c, cancel\_order(c, p, prod\_num))))\end{aligned} \tag{19}$$

Rule 20 shows the mapping 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.

$$\begin{pmatrix} \alpha_0 \,\&\, \mathtt{not}(\alpha_1 \,\&\, (T_1 > T_0)) \,\& \\ current\_date(T_n) \,\& \\ W = trunc((T_n - T_0)/(6048 * 10^5)) \,\& \\ (W < 52) \,\&\, \mathtt{not}(\alpha_2) \,\& \\ T_{end\_w} = T_0 + (W + 1) * (6048 * 10^5)) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \mathtt{add}(\alpha_3), \\ \mathtt{add}(T_i < T_{end\_w}) \end{pmatrix}$$

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

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 and we do not store the sequence of states leading to the present state i.e., the history we cannot reason over the past of any formulae. We can only do it using 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$^\pm$ rule: $\mathsf{X}^-\phi$ is intuitively equivalent to $ctr(S, W, T_s) \,\&\, \phi(T_0) \,\&\, (T_0 = T_s - 1)$. Moreover, we can also translate the U^+ operator when it is used in the *RHS* of the hyMITL$^\pm$ rule: $\phi\ \mathsf{U}^+\psi$ is roughly equivalent to $\psi \rightsquigarrow \mathtt{del}(\phi)$. 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. For instance, $\mathsf{F}^+\phi$ can be achieved if $\mathtt{add}(\phi)$ 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 [41] 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:

$$\begin{pmatrix} \mathbf{H}(request(B, A, P, D, T_r)) \,\& \\ \mathbf{H}(accept(B, A, P, D, T_a)) \,\& \\ T_r < T_a \end{pmatrix} \rightarrow \mathbf{E}(do(A, B, P, D, T_d)) : T_d < T_a + \tau$$

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 mapping of SICs is based on translating events (**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:

$$\begin{pmatrix} utt(D, W_0, request(B, R, A, R', P, T_r)) \,\& \\ utt(D, W_1, accept(A, R', B, R, P, T_a)) \,\& \\ (T_r < T_a) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \mathtt{add}(obl(D, W_2, inform(A, R', B, R, P, T_d))), \\ \mathtt{add}(T_d < T_a + \tau) \end{pmatrix} \tag{20}$$

Although syntactically their language is very similar to ours, they are semantically different. Differently 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 is offered.

6.6 Object constraint language

The work in [10] proposes the Object Constraint Language (OCL) for the specification of artificial institutions. The expression below shows an example of a norm written in OCL:

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

This norm commits the auctioneer to not declare a price lower than the agreed reservation price. As shown in Sect. 5.4, we can also express (rule 18) the case that the auctioneer is obliged to withdraw the good when the call price becomes lower than the reservation price. However, we cannot perform an exhaustive analysis of the language of [10] because neither the syntax nor the semantics are made explicit.

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., [42–44]) or analyse the emergence of norms in multi-agent systems (e.g., [7,45]). On the other hand, logic-oriented contributions focus on the deontic logics required to model normative modalities along with their paradoxes (e.g., [46–48]). 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., [3,49,36,50]).

Vázquez-Salceda et al. [36] 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 [36]: 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. [50] propose the use of norms partially written in OCL, the Object Constraint Language which is part of UML (Unified Modelling Language) [51]. 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. [52] 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. [50]. 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. [8] 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 [8] as a fluent since they assume that every action not permitted is forbidden by default. Although event calculus models time, their deontic fluents do not seem expressive 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 [53] (previous to the work of Artikis et al. [8]), 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 a representation of time that could be easily processed by agents.

Michael et al. [12] 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 [54] 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 [55,56] using the Java programming language [57] 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 [41] 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 [11] reports on the translation of the normative language presented in [36] into Jess rules [58] to monitor and enforce norms. This language captures the deontic notions of permission, prohibition and obligation in several cases such as 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 [59] 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 [59] constraints can only be used to depict the left-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 [20] 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 Sect. 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 Sects. 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 other normative languages proposed with more theoretical concerns in mind to become executable. Our language is rule-based thus addressing the requirement laid out in Sect. 2 that norm-oriented languages should be declarative.

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 [14]. 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.

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.

Providing a computational realisation to abstract models of normative systems is a challenging task. We suggest that if a declarative and compact formalism such as our constraint rule-based language is used, this task could be made easier than, for instance, using a procedural and verbose language such as Java or C++. When mapping alternative formalisms to our rule-based language (as done in Sect. 6), a steep learning curve was required to get familiarised with technical details of the semantics and then how these could be captured with our rules. One way to approach this mapping is to consider how models of normative systems capture commonly occurring phenomena such as norms with deadlines, sanctions and rewards, and so on, and then use rule templates aimed at capturing the same phenomena.

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 Sect. 5.4), we would like to generalise our language to cope with arbitrary actions, as this would allow us to address a larger class of MASs. We would also like to extend the syntax and semantics of our language to support temporal operators for the explicit management of time.

Support can be provided when rules are being designed. We envisage a spectrum of possibilities, ranging from rule templates that can be offered as guidelines, to checking rules for desirable properties (e.g., norms only refer to components of the associated protocols). However, we are aware that we are proposing a formalism with which engineers can program, and ideally a usability analysis should be carried out to investigate how easy-to-use our language is although different dialects and presentations could be custom-built for particular groups of designers.

We envisage two typical ways of using our language: (i) using it directly, either to supplement a MAS with normative regulation or to declaratively implement electronic Institutions, as shown in this paper or; (ii) specifying norms with a language like the one presented in [36] and then using a compiler to translate it into our language to execute it. As a proof of concept, we are implementing one such translator following the principles sketched in Sect. 6.1.

We report our ongoing efforts to incorporate our rule language and its mechanisms to the electronic institutions Development Environment (EIDE)[3]. Norms, in the form of rules, can now be added to EI specifications prepared with the ISLANDER editor and passed to the AMELI middleware which loads them into our rule engine. Whenever agents declare their attempts, AMELI checks the specification if the attempts are meaningful and executes our rule engine to perform further checking and modification of agents' attributes.

We also want to investigate the verification of norms (along the lines of our work in [60]) 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

[3] http://e-institutions.iiia.csic.es/software.html.

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 funded by IEA (TIN2006-15662-C02-01), OK (IST-4-027253-STP), eREP (EC-FP6-CIT5-28575) and Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010).

References

1. Jennings, N. R., Sycara, K., & Wooldridge, M. (1998). A roadmap of agent research and development. *Journal of Agents and Multi-Agents Systems, 1*, 7–38.
2. Axelrod, R. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton studies in complexity. New Jersey: Princeton University.
3. Dignum, F. (1999). Autonomous agents with norms. *Artificial Intelligence and Law, 7*(1), 69–79.
4. López y López, F. (2003). Social power and norms: Impact on agent behaviour. PhD thesis, University of Southampton.
5. Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester, UK: Wiley.
6. Sergot, M. (2001). A computational theory of normative positions. *ACM Transactions on Computational Logic, 2*(4), 581–622.
7. Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial Intelligence, 73*(1–2), 231–252.
8. Artikis, A., Kamara, L., Pitt, J., & Sergot, M. (2005). A protocol for resource sharing in norm-governed Ad Hoc networks. In *Declarative agent languages and technologies II* (Vol. 3476 of LNCS). Springer-Verlag.
9. Cranefield, S. (2005). A rule language for modelling and monitoring social expectations in multi-agent systems. Technical Report 2005/01, University of Otago.
10. Fornara, N., Viganò, F., & Colombetti, M. (2005). An event driven approach to norms in artificial institutions. In *AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM)*. Utrecht.
11. García-Camino, A., Noriega, P., & Rodríguez-Aguilar, J.A. (2005). Implementing norms in electronic institutions. In *Proceedings of 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)* (pp. 667–673). Utrecht, The Nederlands.
12. Michael, L., Parkes, D. C., & Pfeffer, A. (2004). Specifying and monitoring market mechanisms using rights and obligations. In *Proceedings AAMAS Workshop on Agent Mediated Electronic Commerce (AMEC VI)*. New York, USA.
13. d'Inverno, M., & Luck, M. (1998). Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation, 8*(3), 1–27.
14. Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence, 60*(1), 51–92.
15. Searle, J. (1969). *Speech acts, an essay in the philosophy of language*. Cambridge University Press.
16. Jaffar, J., & Maher, M. J. (1994). Constraint logic programming: A survey. *Journal of Logic Programming, 19/20*, 503–581.
17. Mariott, K., & Stuckey, P. J. (1998). *Programming with constraints: An introduction*. USA: MIT Press.
18. Noriega, P. (1997). Agent-mediated auctions: The fishmarket metaphor. PhD thesis, Universitat Autonoma de Barcelona (1997) Number 8 in IIIA Monograph Series.
19. Rodríguez-Aguilar, J. A. (2001). On the design and construction of agent-mediated electronic institutions. PhD thesis, Universitat Autonoma de Barcelona (2001) Number 14 in IIIA Monograph Series.
20. Esteva, M. (2003). Electronic Institutions: from specification to development. PhD thesis, Universitat Politecnica de Catalunya (2003) Number 19 in IIIA Monograph Series.
21. Hübner, J. F., Sichman, J. S., & Boissier, O. (2005). S-MOISE+: A middleware for developing organised multi-agent systems. In *Proceedings of the International Workshop on Organizations in Multi-Agent Systems: From Organizations to Organization-Oriented Programming (OOOP'05)*. Utrecht, The Netherlands.
22. McCallum, M. (2006). MOCHA: Modelling organisational change using agents. PhD thesis, Department of Computing Science, University of Aberdeen, Aberdeen, United Kingdom.
23. Apt, K. R. (1997). *From logic programming to Prolog*. UK: Prentice-Hall.
24. García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., & Vasconcelos, W. (2006). A distributed architecture for norm-aware agent societies. In M. Baldoni et al. (Eds.), *Declarative agent languages and technologies III* (Vol. 3904 of Lecture Notes in Artificial Intelligence (LNAI) pp. 89–105). Berlin Heidelberg: Springer.

25. García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., & Vasconcelos, W. (2006). A rule-based approach to norm-oriented programming of electronic institutions. *ACM SIGecom Exchanges, 5*(5), 33–40.
26. García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., & Vasconcelos, W. (2006). Norm oriented programming of electronic institutions. In *Proceedings of 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*.
27. García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., & Vasconcelos, W. (2006). Norm-oriented programming of electronic institutions: A rule-based pproach. In *Coordination, Organization, Institutions and Norms in agent systems (COIN'06) in 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*.
28. Fitting, M. (1990). *First-order logic and utomated theorem proving*. New York, USA: Springer-Verlag.
29. Tsang, E. P. K. (1993). *Foundations of constraint atisfaction*. Academic Press. Available at http://www.bracil.net/edward/FCS.html.
30. Kramer, B., & Mylopoulos, J. (1992). Knowledge representation. In S. C. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence* (Vol. 1, pp. 743–759). Wiley & Sons.
31. Swedish Institute of Computer Science: SICStus Prolog. (2006). http://www.sics.se/sicstus, viewed on 10 Feb 2006 at 18.16 GMT.
32. North, D. C. (1990). *Institutions, institutional change and economics performance*. Cambridge University Press.
33. Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., & Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)* (pp. 236–243). ACM (2004)
34. Cuní, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C., & Solchaga, T. (2004). MASFIT: Multi-agent system for fish trading. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)* (pp. 710–714). Valencia, Spain (2004).
35. Vasconcelos, W. W., Robertson, D., Sierra, C., Esteva, M., Sabater, J., & Wooldridge, M. (2004). Rapid prototyping of large multi-agent systems through logic programming. *Annals of Mathematics and Artificial Intelligence, 41*(2–4), 135–169.
36. Vázquez-Salceda, J., Aldewereld, H., & Dignum, F. (2004). Implementing norms in multiagent systems. In *Multiagent System Technologies: Second German Conference, MATES 2004* (Vol. 3187 of LNAI., pp. 313–327). Erfurt, Germany: Springer-Verlag (2004).
37. Dignum, V., Meyer, J. J., Dignum, F., & Weigand, H. (2002). Formal specification of interaction in agent societies. In *2nd Goddard Workshop on Formal Approaches to Agent-Based Systems*. Maryland (2002).
38. Hughes, G. E., & Cresswell, M. J. (1996). *A new introduction to modal logic*. Routledge.
39. López y López, F., Luck, M., & d'Inverno, M. (2002). Constraining autonomy through norms. In *Proceedings 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)* (pp. 674–681). Bologna, Italy: ACM Press.
40. Kowalski, R. A., & Sergot, M. J. (1986). A logic-based calculus of events. *New Generation Computing, 4*(1), 67–96. Reprinted in C. Thanos & J. W. Schmidt (Eds.) (1989). *Foundations of Knowledge Based Management Systems* (pp. 23–53). Heidelberg: Springer-Verlag.
41. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2003). Specification and verification of agent interactions using integrity social constraints. Technical Report DEIS-LIA-006-03, Università degli Studi di Bologna (2003).
42. Conte, R., & Castelfranchi, C. (1995). Understanding the functions of norms in social groups through simulation. In N. Gilbert & R. Conte (Eds.), *Artificial Societies. The Computer Simulation of Social Life* (pp. 252–267). London: UCL Press.
43. Conte, R., & Castelfranchi, C. (1993). Norms as mental objects: From normative beliefs to normative goals. In *Proceedings of MAAMAW'93*. Neuchatel, Switzerland (1993).
44. Tuomela, R., & Bonnevier-Tuomela, M. (1995). Norms and agreement. *European Journal of Law, Philosophy and Computer Science, 5*, 41–46.
45. Walker, A., & Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. In *Proceedings International Joint Conference on Multi-Agent Systems (ICMAS)* (pp. 384–389). San Francisco, USA.
46. von Wright, G. H. (1963). *Norm and action: A logical inquiry*. London: Routledge and Kegan Paul.
47. Alchourron, C. E., & Bulygin, E. (1981). The expressive conception of norms. In R. Hilpinen (Ed.). *New Studies in Deontic Logics* (pp. 95–124). London: D. Reidel.
48. Lomuscio, A., & Nute, D. (Eds.). (2004). In *Proceedings of the 7th International Workshop on Deontic Logic in Computer Science (DEON'04)* (Vol. 3065 of Lecture Notes in Artificial Intelligence). Springer-Verlag.
49. Boella, G., & van der Torre, L. (2003). Permission and obligations in hierarchical normative systems. In *Proceedings 8th International Conference in AI & Law (ICAIL'03)*. ACM: Edinburgh.

50. Fornara, N., Viganò, F., & Colombetti, M. (2004). A communicative act library in the context of artificial institutions. In *2nd European Workshop on Multi-Agent Systems* (pp. 223–234). Barcelona.
51. OMG (2005). Unified modelling language. http://www.uml.org.
52. López y López, F., & Luck, M. (2004). A model of normative multi-agent systems and dynamic relationships. In *Regulated agent-based social systems* (Vol. 2934 of LNAI, pp. 259–280). Springer-Verlag.
53. Stratulat, T., Clérin-Debart, F., & Enjalbert, P. (2001). Norms and time in agent-based systems. In *Proceedings 8th International Conference on AI & Law (ICAIL'01)* (pp. 178–185). St. Louis, Missouri, USA.
54. Kollingbaum, M. J. (2005). Norm-governed practical reasoning agents. PhD thesis, Department of Computing Science, University of Aberdeen, United Kingdom.
55. Kollingbaum, M. J., & Norman, T. J. (2003). NoA—a normative agent architecture. In *Proceedings 18th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1465–1466). Acapulco, Mexico: AAAI Press.
56. Kollingbaum, M. J., & Norman, T. J. (2003). Norm adoption in the NoA agent architecture. In *Proceedings 2nd International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'03)*. Melbourne, Australia, USA: ACM.
57. Gosling, J. (1996). *The Java programming Language*. Reading: Addison-Wesley.
58. Sandia Nat'l Labs: Jess. The Rule Engine for Java. (2006) http://www.jessrules.com, viewed on 15 Mar 2006 at 17.50 GMT.
59. Lopes Cardoso, H., & Oliveira, E. (2005). 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)* (Vol. 3690 of LNAI., pp. 256–265). Springer-Verlag.
60. Vasconcelos, W. W. (2004). Norm verification and analysis of electronic institutions. In *Declarative agent languages and technologies II* (Vol. 3476 of LNAI). New York, USA: Springer-Verlag.