Orwell's Nightmare for Agents? Programming Multi-agent Organisations

Nick A.M. Tinnemeier, Mehdi Dastani, and John-Jules Ch. Meyer

Utrecht University, The Netherlands

Abstract. This paper presents a programming language that is designed to implement multi-agent organisations. These organisations are developed as separate entities regulating the behaviour of individual agents that interact with the organisation. The focus is on the normative aspect of organisations that are specified in terms of norms being enforced by monitoring, regimenting and sanctioning mechanisms.

1 Introduction

One of the greatest challenges in the development of multi-agent systems (MAS) is to cope with the large complexity that is caused by the interaction between agents that autonomously pursue their *own* goals. Using an *organisation-centred* approach (as opposed to an *agent-centred* one) is conceived as a way to decrease this complexity and make the development of MAS easier to manage [1,2]. In this approach the organisation is developed as separate entity in terms of organisational concepts (e.g. roles, groups, and norms). From the organisation perspective, the internal state of the agents is not observable, only the actions to be performed in external environments are. That the advantage of using an organisation-centred approach is widely recognised is displayed by the numerous agent methodologies (e.g. Gaia [3]), models (e.g. OperA [4], $\mathcal{M}$oise$^+$ [5], E-Institutions [6], AGR [7]) and frameworks (e.g. AMELI [8], $\mathcal{S}$-$\mathcal{M}$oise$^+$ [5]) that adopt (at least some) organisational concepts.

Normative elements constitute an important aspect in developing organisations. Since no assumptions are made about the inner workings of agents participating in the organisation, also no assumptions can be made about the behaviour they will exhibit. Norms provide the agents with some behavioural guideline, thereby ensuring that the objectives of the organisation are not endangered. Normative elements thus specify the rules of behaviour (norms) the agents ought to obey when participating in the organisation. These norms are often specified by means of concepts like permissions, obligations, and prohibitions.

To fully exploit the results of MAS research in the development of complex software systems, sophisticated programming languages are needed to put theory to practice. Such a programming language must satisfy the fundamental requirement that it is *"reasonably natural for solving problems, at least problems within its intended application area"* [9]. A language is considered reasonably natural if

K.V. Hindriks, A. Pokahr, and S. Sardina (Eds.): ProMAS 2008, LNAI 5442, pp. 56–71, 2009.

we are able to keep the concepts used in analysis and design alive in the implementation. Without this, the concepts used in specification and design need to be implemented in an ad-hoc manner, making the implementation a burdensome task prone to errors, having negative impact on the product's maintainability.

Although we acknowledge that a multi-agent organisation encompasses more than the normative element only (see section 2), the focus of this paper is on the normative element of multi-agent organisations. We aim at operationalization of the normative elements based on which we design a programming language to implement organisations. In the literature there is related work on electronic institutions for regulating agents according to certain norms. In particular, ISLANDER[6] is a formal framework for specifying norms in institutions, and is used in the AMELI platform[8] for executing electronic institutions based on rules provided in it. A difference with our work is that ISLANDER/AMELI is primarily aimed at very concrete norms in the form of procedures, typically in the form of "ought-to-do's" on speech acts, while in our approach we are primarily concerned with more abstract, declarative norms ("ought-to-be's", cf. [10,11]). Another difference is that in contrast to our approach, in ISLANDER/AMELI norms can never be violated by agents.

Another work that is concerned with the operationalization of multi-agent systems using organisational concepts is the work of $\mathcal{S}$-$\mathcal{M}$oise$^+$[5]. $\mathcal{S}$-$\mathcal{M}$oise$^+$ is an organisational middleware, in which also more abstract, declarative norms are used. In this approach, however, norms merely serve as guidelines of proper behavior for the agents in the system. It lacks a mechanism to detect whether an agent has actually fulfilled its obligations, let alone a sanctioning mechanism. In our approach, however, programming MAS means to program detection and sanctioning mechanisms since these determine the type of coordination put into place by the system.

A different approach of regulating the external behavior of individual agents is that of using coordination artefacts [12]. Just like these artefacts, we propose to develop an organisation as a separate entity with the goal of coordinating external agents. In our work, however, coordination is achieved by means of high-level constructs such as norms, more closely relating to the models in which MAS are usually designed, and not so much as low-level coordination concepts such as synchronization of processes coordination artefacts are based on.

Closely related to this work is [13] in which a simplified version of a programming language that is designed to implement norm-based artefacts is proposed, along with a logic that can be used to specify and verify properties of programs developed in this language. In this paper the programming language is enriched with temporal aspects, constructs to refer to actions and roles. A logic to reason about programs implemented in our language is omitted, however.

Section 2 explains the key concepts of normative MAS and organisations reasearch that are of importance in this paper. Section 3 presents the syntax and semantics of our programming language. Section 4 explains how normative multi-agent organisations can be implemented by this language, and section 5 concludes this paper and hints at some directions for future research.

2 Key Concepts of Multi-agent Organisations

Our approach of designing a normative multi-agent programming language is based on ideas from research in normative systems and research in multi-agent organisations. Therefore, this section introduces some of the key concepts that we use from this area.

In general, the organisations which we aim to implement with our programming language consist of the following components. Part of the organisation is the *environment* which state can be modified by the external actions of *agents* that interact with the organisation. No assumptions are made about the inner workings of these agents. We assume that the organisation is able to determine the effects of these external actions. Also part of the organisation is a *detectioning mechanism* that normatively assesses the organisation and a *sanctioning mechanism* imposing sanctions as a result of this normative assessment. The organisation thus becomes a Big Brother monitor the agents' behaviour and imposing sanctions accordingly, thereby embodying Orwell's nightmare for agents. It should be emphasised that in our approach the organisation is a passive entity merely reacting to the actions that external agents perform within the environment.

To illustrate these components as well as some other concepts that are of importance in our approach, we use a simple example of a software simulation of a railway system. In this simulation software agents play the role of passengers that travel by train. The conditions of using the transport system are captured as norms. Other examples of applications we are targeting at are, for instance, a financial administration database, a conference management system, or an online marketplace.

2.1 The Normative Aspect of Organisations

Norms often find their representation in deontic logic, a logic for reasoning about ideal and actual behaviour. Many different deontic logics have been developed introducing operators for permission, obligation and prohibition (see [14] for an overview). In this work norms are represented as elementary counts-as statements as motivated and developed in [15].

Counts-as statements are used to classify or make a judgment about the organisation. Herewith, a distinction is made between between brute and normative/institutional facts as first advanced in [16]. The environment is described by means of brute facts, e.g. *"agent a is in the train without a valid ticket"*. The value judgment of this situation is expressed by means of normative facts, e.g. a classification as good (desirable), ugly (undesirable, but tolerated) or bad (extremely undiserable, and not tolerated). For instance, to say that for agent i it is forbidden to be in the train without a ticket is to say in terms of counts-as statements that being in the train without a ticket counts as a violation for i:

$$\texttt{in\_train}(i) \wedge -\texttt{ticket}(i) \Rightarrow \texttt{viol}(i) \tag{1}$$

Counts-as statements thus label the situation as expressed by brute facts with normative facts, thereby normatively assessing the organisation.

Deontic notions like prohobition, obligation, and permission can be expressed as counts-as statements. Intuitively (see [15] for a thorough analysis), the deontic notion of being prohibited to be in a (brute or normative) state characterised by p can be modelled in terms of counts-as statements by stating that p counts-as a violation. If it is permitted to be in a situation in which p holds then being in a situation in which p holds does not necessarily count as a violation[1]. An obligation to be in a (brute or normative) state in which p holds means that being in a state in which p does *not* hold necessarily counts as a violation.

To validate the norm described by the counts-as statement as specified by formula 1 only the current state of the system needs to be judged, because the act of being in the train without a ticket can be detected at the very moment the agent is in the train without a ticket. There are also norms, however, which need a time line to be validated. For instance, in order to validate a norm like: *"A passenger ought to buy a ticket while on the train."*, requires an assessment of the whole period in which the passenger travels by train. To be able to also express this kind of norms, which have temporal character, we enrich the language of counts-as statements with temporal operators.

In order to motivate the agents that participate in the organisation to obey the rules of behaviour, besides a representation of the norms also a mechanism is needed for letting the agents abide by the norms. One way of assuring that the agents comply with the norms is to *rule out* all the actions that will lead to a violation state, such that a violation will never happen. This way of carrying out the norms is referred to as regimentation. The organisation can somehow prevent an agent from performing an external action that causes a violation. This presupposes the organisation to have the ability to determine the effect of the actions that can be performed by the agents. An example of a case in which this presumption holds is an operating system that can disable certain operations for users that do not have the right permissions. Note that this presumption does not imply that the system has control over the internals of the agent, it can still try to perform the operation, but the result is simply not effectuated by the operating system.

As alternative to regimentation an enforcement mechanism can be used. Enforcement is based on the idea of responding after a violation of the norms has occurred. Following the old Roman saying "ubi lex ibi poena" (where there is a law, there is a sanction), we also define rules that specify the sanction that should be imposed as a consequence of this violation. For example, the sanction belonging to the violation caused by agent i travelling without a ticket is a fine of 10 credits, written as:

$$\texttt{viol}(i) \Rightarrow \texttt{fined10}(i) \tag{2}$$

[1] In [15] also a more strong notion of obligation is defined in which p is permitted iff p necessarily counts as $-viol$. In this paper, we use the weak notion, however.

These rules are in fact the inversion of the counts-as rules. Instead of stating which normative terms apply as a consequence of brute facts, sanctioning rules associate new brute facts with certain normative facts. An enforcement mechanism is especially useful in case the system cannot determine the effects of certain actions. However, even when the organisation is able to apply regimentation, enforcement might still be fruitful, because allowing for violations contributes to the flexibility and autonomy of the agent's behaviour [17].

2.2 Other Organisational Aspects

As already mentioned, an organisation consists of more than normative aspects only. Besides the normative aspects, in [18] three other major organisational aspects were identified. *Functional elements* refer to the functioning of the organisation by stating its main objectives, and how they can be achieved. For instance, by specifying global plans that prescribe the steps that should be taken to reach the objectives (cf. functional specification of $\mathcal{M}$oise$^+$ [5]). *Structural elements* define the specific structure of the organisation that is used to reach these objectives, and is usually defined by means of the roles that should be fulfilled along with the relations between these roles, such as power, coordination, and control (cf. [15]). *Dialogical elements* deal with the communicative aspects of the organisation ensuring efficient communication between agents, an important prerequisite in reaching the organisational objectives. They specify, for instance, communication protocols (cf. [6]) specifying the possible dialogic interaction between roles.

Roles form an important concept in all organisational aspects. In [19] a role is described as "*...a class that defines a normative behavioral repertoire of an agent.*". In this work we will treat roles as being labels denoting the name of the roles agents can play within an organisation. Special facts $\texttt{rea}(i, \rho)$ then model that agent i has enacted a role typified by the label ρ. Moreover, we will introduce actions for enacting and de-enacting (deact from now on) roles that allow agents to enact and/or deact roles dynamically. Later on, we will show that this simple view on roles in combination with the normative aspect of the organisation allows to already handle some structural aspects of an organisation.

3 A Normative Multi-agent Programming Language

This section presents the relevant parts of the syntax and semantics of a programming language that is designed to implement organisations.

3.1 Syntax of Programming Language

Agents that interact with the organisation can perform actions to change the organisation. In particular, these actions allow agents to change the environment (brute state of the organisation), to enact and deact roles, and to communicate with each other. In defining the action language (and in the following), we assume

a set *RoleName* with typical element ρ as the set of labels identifying the roles that agents can play within the organisation, and we assume a set of agents that will be uniquely identified by $i, j, \ldots$

Definition 1 (actions). *Let $\rho \in RoleName$, let $ExtAct$ with typical element α be the set of external actions, and let $ComAct$ be the set of communicative actions with typical element γ_j in which γ is the identifier of the illocutionary act, and j the identifier of the receiving agent. Then the set of actions Act with typical element β is defined as:*

$$Act = ExtAct \cup ComAct \cup \{enact(\rho), deact(\rho)\}$$

The state of the organisation is built of brute facts specifying the environment and normative facts specifying the judgment about the organisation. This same distinction between brute and normative facts is made for the logical language for expressing the facts representing the organisation: brute facts are modelled in the propositional language L_b, whereas normative facts are modelled in the propositional language L_n. Although a first-order language is much more expressive, in this paper a propositional one is used for the sake of readability.

The special propositions of the form $\texttt{rea}(i, \rho)$ are used to model the fact that agent i has enacted role ρ. Propositions of the form $\texttt{done}(i, \beta)$ are used to denote that agent i has *just* performed action β. This allows to refer to actions in expressing the norms. The special proposition $\texttt{viol}_\bot$ is used to mark those situations that are so undesirable that they are *strongly* forbidden in the sense that the system assures that never such a state is reached. These propositions thus pertain to the norms that are to be regimented.

The normative properties that are used for an assessment of the organisation are expressed in L, a language of propositional linear time logic (PLTL) (see [20] for an introduction). Norms can thus have a temporal character. In particular, the operators X (neXt), G (Globally), F (Eventually), and U (Until) are introduced. Some norms might also refer to normative facts, for instance, a violation an agent has committed at some moment in the past. Therefore, the language for expressing the norms can range over both brute and normative facts.

Definition 2 (logical languages). *Given the set of atomic propositions P, special propositions $\texttt{done}(i, \beta)$ for all $\beta \in Act$ and all agents i, special propositions $\texttt{rea}(i, \rho)$ for all $\rho \in RoleName$ and all agents i, the language L (norms), L_b (brute), and L_n (normative) are defined as:*

- *if $p \in P$ then $p, -p \in L_b$*
- $\texttt{done}(i, \beta) \in L_b$
- *if $q \in P$ then $q, -q \in L_n$*
- $\texttt{rea}(i, \rho) \in L_n$
- $\texttt{viol}_\bot \in L_n$
- *if $\phi_1, \phi_2 \in (L_b \cup L_n)$ then $\phi_1, \phi_2 \in L$*
- *if $\phi_1, \phi_2 \in L$ then $\phi_1 \wedge \phi_2, \neg\phi_1 \in L$*
- *if $\varphi_1, \varphi_2 \in L$ then $\mathrm{X}\varphi_1, \mathrm{G}\varphi_1, \mathrm{F}\varphi_1, \varphi_1 \mathrm{U} \varphi_2 \in L$*

It is assumed that the brute and normative language are mutually exclusive, more formally $L_b \cap L_n = \emptyset$.

The performance of external actions by agents changes the state of the environment. Brute effects specify how the organisation can advance in its computation by stating which brute facts are changed after the execution of the external action, i.e. they determine the brute effect of the action execution. The specific effect of performing an external action depends on the current state of the environment. A brute effect is thus a triple consisting of a pre-condition specifying when the action can be executed, the action name that is to be executed, and a post-condition listing the brute facts that hold after execution.

Definition 3 (brute effects). *Let ExtAct be the set of external actions and ComAct be the set of communicative actions an agent can perform. The set of brute effects $\mathcal{R}_b$ is defined in the following manner:*

$$\mathcal{R}_b = \{(p_1, \ldots, p_k)\ \beta\ (p_{k+1}, \ldots, p_n) \mid p_1, \ldots, p_n \in L_b \text{ and } \beta \in (ExtAct)\}$$

Normative rules are used to normatively assess the organisation. Recall that norms are expressed as elementary counts-as rules associating normative facts with a certain situation the organisation is in. This situation is described by the antecedent by means of a temporal formula ranging over brute and normative facts. The consequent then specifies which normative facts are to be associated with this situation.

Definition 4 (normative rules). *The set of normative rules $\mathcal{R}_n$ is defined as follows:*

$$\mathcal{R}_n = \{(\varphi_1, \ldots, \varphi_n) \Rightarrow (q_1, \ldots, q_m) \mid \varphi_1, \ldots, \varphi_n \in L \text{ and } q_1, \ldots, q_m \in L_n\}$$

It is possible that the system ends up in a less desirable state, for instance, because some agent violated a norm. Sanctioning rules can then be used to indicate the punishments that are imposed as consequence of this violation. This mechanism thus pertains to enforcement of the norms. It should be noted that the verdicts raised by the normative rules are not necessarily always of an unfavorable nature. Sanctions can thus either be positive (rewards) or negative (punishments). Just like normative rules, sanctioning rules have an antecedent and a consequent, with the antecedent referring to the normative judgment of a particular state and the consequent being the sanction that should be imposed.

Definition 5 (sanctioning rules). *The set of sanction rules $\mathcal{R}_s$ is defined as:*

$$\mathcal{R}_s = \{(q_1, \ldots, q_n) \Rightarrow (p_1, \ldots, p_m) \mid q_1, \ldots, q_n \in L_n \text{ and } p_1, \ldots, p_m \in L_b\}$$

Note that the antecedent of a sanctioning rule refers to the normative judgment about a state and can only contain normative facts. The intuitive meaning is that given a normative judgment of a certain state the consequent states the sanction in terms of brute facts that are to be imposed on this state.

In the following, for each normative or sanctioning rule r we refer to its condition by $cond(r)$, and to its consequence by $cons(r)$.

3.2 Semantics of Programming Language

Having defined the syntax for specifying an organisation, next we define the operational semantics by means of a transition system [21]. Each transition corresponds to a single computation step describing the transformation of one configuration (program state) into another. Before defining the notion of organisational configuration, we first define the notion of organisational state and history and some necessary functions operating on them.

As explained before, an organisation is characterised by brute facts and normative facts. An organisational state, describing the state of the organisation at a certain moment, is therefore defined as a tuple consisting of a set of brute and a set of normative facts. Seeing that the organisational state evolves due to the execution of actions by agents, and the application of normative and sanctioning rules, we define a consistency preserving operator for updating the organisational state.

Definition 6 (organisational state). *Given a consistent set of brute facts $\mathcal{B} \subseteq L_b$ and a consistent set of normative facts $\mathcal{N} \subseteq L_n$, an organisational state Ω is defined as a tuple $\langle \mathcal{B}, \mathcal{N} \rangle$.*

Further let $\overline{X}$, the complement of a set of (brute or normative) facts X be the set $\{\phi \mid -\phi \in X\} \cup \{-\phi \mid \phi \in X\}$, then the functions $\oplus_b$ and $\oplus_n$ for updating an organisational state are defined as:

$$\begin{aligned} \langle \mathcal{B}, \mathcal{N} \rangle \oplus_b X_b &= \langle X_b \cup \{\mathcal{B} \setminus \overline{X}_b\}, \mathcal{N} \rangle \\ \langle \mathcal{B}, \mathcal{N} \rangle \oplus_n X_n &= \langle \mathcal{B}, X_n \cup \{\mathcal{N} \setminus \overline{X}_n\} \rangle \end{aligned}$$

The performance of actions by agents changes the organisational state. The brute effects are used in determining the effects of external action performance. Not only external actions change the organisational state, however. For example, when an agent i performs an `enact` action the agent has enacted a role ρ, which is being modelled by the normative fact $\texttt{rea}(i, \rho)$. Given an organisational state, an action and an agent i, we define the effects function *effect* in order to determine the new organisational state as a consequence of the performance of the action by agent i.

Definition 7 (effects function). *Given brute effect $b = (\Phi\ \alpha\ \Psi)$, organisational states $\Omega = \langle \mathcal{B}, \mathcal{N} \rangle$ and $\Omega' = \langle \mathcal{B}', \mathcal{N}' \rangle$ such that $\mathcal{B}' = \mathcal{B} \setminus \{\texttt{done}(i, \beta) \mid \beta \in Act\}$ and such that $\mathcal{N}' = \{\texttt{rea}(i, \rho) \mid \texttt{rea}(i, \rho) \in \mathcal{N}\}$, functions $effect(i, \beta, \Omega)$ and $effect(i, b, \Omega)$ determine the effect of the performance of β or application of b (corresponding to performance of α) in organisational state Ω by agent i:*

$$\begin{aligned} effect(i, b, \Omega) &= \Omega' \oplus_b (\{\texttt{done}(i, \alpha)\} \cup \Psi) \\ effect(i, enact(\rho), \Omega) &= (\Omega' \oplus_b \{\texttt{done}(i, enact(\rho))\}) \oplus_n \{\texttt{rea}(i, \rho)\} \\ effect(i, deact(\rho), \Omega) &= (\Omega' \oplus_b \{\texttt{done}(i, deact(\rho))\}) \oplus_n \{-\texttt{rea}(i, \rho)\} \\ effect(i, \gamma_j, \Omega) &= \Omega' \oplus_b \{\texttt{done}(i, \gamma_j)\} \end{aligned}$$

After the performance of each action, the brute state is updated with the fact that the agent has performed that action. In particular, the fact $\texttt{done}(i, \beta)$ is

designed to mean that the previous state has been transformed in a new state, *because* agent i has just performed action β. Therefore any previous `done` fact is removed from $\mathcal{B}$. As we shall see later on, only one agent can perform an action per computation step, conforming to an interleaved action execution strategy.

In each state the brute facts change as a result of the agents' actions, leading to a new state. Normative facts directly depend on the current situation of the system, i.e. a normative assessment needs to be done for each newly reached state. Therefore, in determining the effects of an action, all normative facts of the preceding state ($\mathcal{N}$) are removed in the subsequent state ($\mathcal{N}'$), such that this fresh state can be normatively assessed after the brute effects have been determined. Note that the `rea` propositions are not removed, because once an agent has enacted a role it will keep doing so until it performs a deact.

Recall that with the norms of the organisation we cannot only reason about the present situation, but can also reason about things that happened in the past. Therefore, we also need to remember the situations that occurred in the past, and introduce the concept of an organisational history.

Definition 8 (organisational history). *An (organisational) history σ is defined as a finite trace $\Omega_0\Omega_1\cdots\Omega_n$ with $\Omega_i = \langle\mathcal{B}_i,\mathcal{N}_i\rangle$ being an organisational state for all $i \leq n$. The concatenation operator $\circ$ on traces is defined in the usual way. Moreover, given a history $\sigma = \Omega_0\Omega_1\cdots\Omega_i\cdots\Omega_n$, the suffix of σ from i, denoted as (σ, i), is defined as the history $\Omega_i\cdots\Omega_n$.*

To illustrate the intuitive meaning of a history, consider an organsitational history $\Omega_0\cdots\Omega_n$. Then the first state Ω_0 models the initial state of the organisation. The last state Ω_n then models the most recent state that has been reached due to the performance of an action by some agent. In case a new action is performed by an agent, a new state Ω_{n+1} is added, denoting the new organisational history as consequence of carrying out this action. The whole trace $\Omega_0\cdots\Omega_n$ thus models the present and the past of the organisation at a certain moment. We emphasise that we use histories as snapshots pertaining to the execution thus far, and not so much as possible executions such traces usually pertain to.

The satisfaction relation $\models$ is defined on such organisational histories, making it slightly different from the standard satisfaction relation for PLTL. Firstly, in contrast to the traces on which PLTL formulae are evaluated an organisational history is finite. Secondly, the states a trace is composed of now consist of a tuple of sets of propositions instead of a single set. Since norms often refer to the current situation, the macro *now* is defined to facilitate the notation.

Definition 9 (logical entailment). *Let $\varphi, \psi \in L$ and let $p \in L_b$ and $q \in L_n$. Also let σ be a history of length $n+1$ with $\Omega_i = \langle\mathcal{B}_i,\mathcal{N}_i\rangle$ for each $0 \leq i \leq n$. Then the entailment relation $\models$ w.r.t. trace σ is defined as:*

1) $(\sigma, i) \models (-)p$ iff $(-)p \in \mathcal{B}_i$
2) $(\sigma, i) \models (-)q$ iff $(-)q \in \mathcal{N}_i$
3) $(\sigma, i) \models \neg\varphi$ iff $(\sigma, i) \not\models \varphi$
4) $(\sigma, i) \models \varphi_1 \wedge \varphi_2$ iff $(\sigma, i) \models \varphi_1$ *and* $(\sigma, i) \models \varphi_2$
5) $(\sigma, i) \models \mathrm{X}\varphi$ iff $i < n$ and $(\sigma, i+1) \models \varphi$
6) $(\sigma, i) \models \varphi_1 \mathrm{U} \varphi_2$ iff $\exists\, j \geq i.\ ((\sigma, j) \models \varphi_2$ and $(\forall\, i \leq k < j.\ (\sigma, k) \models \varphi_1))$

The auxiliary operators F *and* G *are defined in terms of the already existing operators, such that* $\mathrm{F}\varphi \equiv \top \mathrm{U}\varphi$ *and* $\mathrm{G}\varphi \equiv \neg \mathrm{F} \neg \varphi$. *Further the macro now for discerning the last state of a trace is defined as:* $now(\varphi) \equiv \mathrm{F}(\varphi \wedge \mathrm{X}\bot)$

Each organisational state Ω that has just been reached needs to be assessed by the normative rules, and sanctions need to be imposed accordingly. This is a matter of adding the consequences of the (normative or sanctioning) rules of which the premisses is satisfied to Ω. For this purpose, we define the applicability of rules given a certain history, and the closure of a last state of a history under a set of rules (being either normative or sanctioning rules). As the premisses of a normative rule is a temporal formula, it needs to be evaluated on the whole history Ω is part of. The premisses of sanctioning rules, on the other hand, refers to the assessment of the most recent state and needs to be evaluated on Ω.

Definition 10 (applicable rules and closure under rules). *Given a set of rules* R *(either* R_n *or* R_s*) and a trace* $\sigma = \Omega_0 \Omega_1 \cdots \Omega_{n-1} \Omega_n$, *the set of applicable rules w.r.t.* σ *is defined as:*

$$\begin{aligned} Appl(R_n, \sigma) &= \{r \mid r \in R_n \text{ and } (\sigma, 0) \models cond(r)\} \\ Appl(R_s, \sigma) &= \{r \mid r \in R_s \text{ and } (\sigma, n) \models cond(r)\} \end{aligned}$$

Let $\oplus$ *be either* $\oplus_b$ *or* $\oplus_n$ *and let*

$$\begin{aligned} Cl_0^R(\sigma, \Omega_n, \oplus) &= \Omega_n \oplus \left(\bigcup_{r \in Appl(R,\sigma)} cons(r)\right) \\ Cl_{k+1}^R(\sigma, \Omega_n, \oplus) &= \Omega'_n \oplus \left(\bigcup_{r \in Appl(R, \Omega_0 \Omega_1 \cdots \Omega_{n-1} \circ \Omega'_n)} cons(r)\right) \\ &\quad \text{s.t. } \Omega'_n = Cl_k^R(\sigma, \Omega_n, \oplus) \end{aligned}$$

then the closure of Ω_n *under* R *is defined as* $Cl_k^R(\sigma, \Omega_n, \oplus)$ *for the minimal* k *such that* $Cl_{k+1}^R(\sigma, \Omega_n, \oplus) = Cl_k^R(\sigma, \Omega_n, \oplus)$.

It should be noted that such a closure only exists under certain conditions for the set of rules R (see [13] for what these conditions are).

Having defined all the necessary ingredients for defining the transition rules, we are now able to define the organisational configuration. An organisational configuration is a tuple composed of a set of agents that act in the organisation and a history, modelling the situation of the organisation.

Definition 11 (organisational configuration). *Let* $\mathcal{A} = \{A_1, \ldots, A_n\}$ *be the set of agents with each* A_i *the configuration of individual agent* i, *and let* σ *be an organizational trace. The configuration of an organization is defined as* $\langle \mathcal{A}, \sigma \rangle$.

Before defining the transition rules specifying how one organisational configuration can evolve into another, we first define possible transitions individual agents can make, without making any assumptions about their configuration.

Definition 12 (agent transitions). *The agent transitions are defined as:*

$$\mathrm{ACT_s} : A_i \xrightarrow{\beta} A'_i \qquad \mathrm{SEND_s} : A_i \xrightarrow{\gamma_j !} A'_i \qquad \mathrm{REC_s} : A_i \xrightarrow{\gamma_i ?} A'_i$$

Transition $\mathrm{ACT_s}$ states that an individual agent is capable of performing external actions, role enactment and deactment. Transition $\mathrm{SEND_s}$ indicates that agents can always perform communicative acts, and $\mathrm{REC_s}$ indicates that the agent can always receive communicative acts sent by other agents.

The actual effect of individual agents' actions is determined by the organisation. The transition rules at the multi-agent level are therefore defined in terms of the single agent transitions, and define what it means to execute an action in the organisation. In particular, the transitions rules defined below specify what happens at the multi-agent level when an agent performs an external action ($\mathrm{EXT_m}$), a deact or an enact ($\mathrm{ROL_m}$), and a communicative action ($\mathrm{COM_m}$). For an external action to be executed the pre-condition of the brute effect specifying the effect should be satisfied by the current state of the organisation. Moreover, the execution of a communicative action synchronises the sender and receiver.

The detection and sanctioning mechanism as discussed in section 2 is the same for all three types of actions. To start with a new state Ω as result of the action execution is determined by means of the effects function. The new situation of the organisation is normatively assessed by closing off the former history σ with Ω appended as last state under the normative rules. State Ω' is the result of this assessment. The sanctions that need to be imposed are determined similarly, that is by closing off Ω' under the sanctioning rules, resulting in Ω''. State Ω'' thus corresponds to the normatively assessed state with sanctions imposed accordingly, the organisation would reach after performance of the action.

Whether the action to be performed is tolerated depends on the normative judgment. That is to say, when Ω'' entails $\mathtt{viol}_\perp$, this means that the organisation would end up in a strongly forbidden situation. In this case the action is blocked, conforming to regimentation. If this is not the case, the organisation can advance in its computation; the history is updated with the new agent configurations and newly reached state Ω''. It should thus be noted that both enforcement and regimentation are captured in each of these separate transitions.

Definition 13 (multi-agent transitions). *Let R_n be the set of normative rules, R_s the set of sanctioning rules, $\alpha \in ExtAct$, $\beta \in \{enact(\rho), deact(\rho)\}$, $\gamma_j \in ComAct$, $b = (\Phi\ \alpha\ \Psi)$ s.t. $b \in \mathcal{R}_b$, and let $\langle \mathcal{A}, \sigma \rangle$ be a multi-agent system with $\sigma = \Omega_0 \ldots \Omega_n$. The multi-agent transitions are defined as:*

$$\mathrm{EXT_m}: \frac{\begin{array}{c} A_i \xrightarrow{\alpha} A_i' \quad \Omega_n \models \Phi \quad \Omega = effect(i, b, \Omega_n) \\ \Omega' = Cl^{R_n}(\sigma \circ \Omega, \Omega, \oplus_n) \quad \Omega'' = Cl^{R_s}(\sigma \circ \Omega', \Omega', \oplus_b) \\ (\sigma \circ \Omega'', n+1) \not\models \mathtt{viol}_\perp \end{array}}{\langle \mathcal{A}, \sigma \rangle \longrightarrow \langle (\mathcal{A} \setminus \{A_i\}) \cup \{A_i'\}, \sigma \circ \Omega'' \rangle}$$

$$\mathrm{ROL_m}: \frac{\begin{array}{c} A_i \xrightarrow{\beta} A_i' \quad \Omega = effect(i, \beta, \Omega_n) \\ \Omega' = Cl^{R_n}(\sigma \circ \Omega, \Omega, \oplus_n) \quad \Omega'' = Cl^{R_s}(\sigma \circ \Omega', \Omega', \oplus_b) \\ (\sigma \circ \Omega'', n+1) \not\models \mathtt{viol}_\perp \end{array}}{\langle \mathcal{A}, \sigma \rangle \longrightarrow \langle (\mathcal{A} \setminus \{A_i\}) \cup \{A_i'\}, \sigma \circ \Omega'' \rangle}$$

$$\text{COM}_\text{m}: \frac{\begin{array}{c} A_i \xrightarrow{\gamma_j!} A_i' \quad A_j \xrightarrow{\gamma_j?} A_j' \quad \Omega = \mathit{effect}(i, \gamma_j, \Omega_n) \\ \Omega' = Cl^{R_n}(\sigma \circ \Omega, \Omega, \oplus_n) \quad \Omega'' = Cl^{R_s}(\sigma \circ \Omega', \Omega', \oplus_b) \\ (\sigma \circ \Omega'', n+1) \not\models \texttt{viol}_\bot \end{array}}{\langle \mathcal{A}, \sigma \rangle \longrightarrow \langle (\mathcal{A} \setminus \{A_i, A_j\}) \cup \{A_i', A_j'\}, \sigma \circ \Omega'' \rangle}$$

Recall that, due to the construction of the effects function (Def. 7) the performance of a communicative action, deact and enact does not change the brute facts of the system except for the addition of the `done` proposition. Tolerated execution of an enact or deact leads to the addition or deletion of a *rea* proposition in the normative facts, such that it is remembered that the agent has or has not enacted a certain role.

4 Implementing Multi-agent Organisations

In this section we show by example how the normative multi-agent organisation programming language can be used to implement multi-agent organisations. To provide a broader view on the intended application area for our language, we do not limit ourselves to the toy example of the railway system.

In the railway simulation agents can be at the platform (being expressed as `at_platform`) or in the train (being expressed as `in_train`). If the agent is at the platform and not in the train, she can enter the train by performing an `embark` action, of which the result is that the agent is in the train and not at the platform anymore. The external actions agents can perform to change the environment are expressed by brute effects. Consider, for example, the brute effect of the `embark` action:

$$(\texttt{at\_platform}(i), -\texttt{in\_train}(i))\ \texttt{embark}\ (-\texttt{at\_platform}(i), \texttt{in\_train}(i))$$

The railway regulations state the rules of behaviour the travellers ought to follow and are expressed by means of the normative rules. Suppose, for example, that passengers are obliged to buy a ticket before entering on the platform. Violating this norm is not considered to be a serious violation and the sanction is a fine of 10 credits. Being in the train without a ticket, however, is a more serious violation of which the sanction is a fine of 50 credits. In our approach these two rules of conduct can be expressed by the normative rules:

$$\begin{aligned} \mathit{now}(\texttt{at\_platform}(i) \wedge -\texttt{ticket}(i)) &\Rightarrow \texttt{viol}^{tp}(i) \\ \mathit{now}(\texttt{in\_train}(i) \wedge -\texttt{ticket}(i)) &\Rightarrow \texttt{viol}^{tt}(i) \end{aligned}$$

Recall that *now* is used to discern the last organisational state of a history. The above rules thus have the intuitive reading: "currently being at the platform (or in the train, respectively) without a ticket counts as a violation". Note the usage of labels *tp* (ticket platform) and *tt* (ticket train) on the violation propositions to match a violation with the norm that has been violated. They are used to discriminate from different violations in defining the sanctioning rules:

$$(\texttt{viol}^{tp}(i)) \;\Rightarrow\; \texttt{fined}10(i)$$
$$(\texttt{viol}^{tt}(i)) \;\Rightarrow\; \texttt{fined}50(i)$$

The norms defined above concern enforcement. To illustrate regimentation, suppose that the railway system allows passengers to violate the norm of travelling with a ticket only once. In other words, if a passenger has been caught travelling without a ticket in the past, then travelling without a ticket for the second time is regimented. This is expressed by means of a normative rule as:

$$\mathrm{F}(\texttt{viol}^{tt}(i) \wedge \mathrm{XF}(\texttt{in\_train}(i) \wedge -\texttt{ticket}(i))) \;\Rightarrow\; \texttt{viol}_{\perp}$$

Transition rule $\mathrm{EXT_m}$ ensures that the organisation will never end up in this situation, because all actions that will lead to $\texttt{viol}_{\perp}$ are blocked. Intuitively, this can be thought of as placing a gate just in front of the entrance of the train that will remain closed in case a passenger tries to embark without a ticket for the second time, making it physically impossible for the agent to enter. Note that since states that are marked by $\texttt{viol}_{\perp}$ will never be reached, no sanctions for $\texttt{viol}_{\perp}$ need to be defined.

To also show how a more complex norm of a temporal nature can be expressed in our normative language, suppose that passengers no longer need to buy a ticket before entering the train, but now should buy their ticket during their train ride. This norm can be expressed as a normative rule in the following manner:

$$\mathrm{F}(\texttt{in\_train}(i) \wedge -\texttt{ticket}(i)\mathrm{U}(-\texttt{in\_train}(i) \wedge \mathrm{X}\perp)) \;\Rightarrow\; \texttt{viol}^{tt}(i)$$

intuitively meaning that agent i is committing an offence when she has not been in possession of a ticket until she got off the train. Note that due to the usage of $\mathrm{X}\perp$ this violation is detected at the very moment the agent leaves the train.

Hitherto the focus has been on programming the normative elements of multi-agent organisations. As already mentioned, an organisation encompasses more than only normative elements. The $rea(i, \rho)$ propositions were merely used to denote the fact that agent i has enacted role ρ, and have not played a very significant part thus far. However, in combination with the normative aspects already some structural aspects of an organisation can be expressed.

As already mentioned in section 2, norms are often associated with a certain role. Consider, for example, a conference management system. Usually, for an agent playing the role of program chair other norms are in effect than for an agent playing the role of author. Then, in our approach a role somehow becomes a means of modularising the normative rules. To illustrate, consider the following normative rule:

$$now(\texttt{rea}(i, \texttt{author}) \wedge \texttt{registration\_closed} \wedge \texttt{done}(i, \texttt{register})) \Rightarrow \texttt{viol}^{reg}$$

expressing that an agent playing the role of author can still register for the conference even if the registration has already been closed. The possible sanction could then be a higher entrance fee. The antecedent of this normative rule will only be satisfied if agent i has currently enacted the role of author, and is thus

only in effect for authors. Note that this normative rule refers to a concrete action instead of a declarative description of the organisation as was the case in the norms before. Of course, the inverse of this approach, stating that a certain norm is applicable for all roles except a certain role could also be taken. To express, for example, that agents not playing the role of author cannot register after the registration has been closed, is to write:

$$now(-\texttt{rea}(i, \texttt{author}) \wedge \texttt{registration\_closed} \wedge \texttt{done}(i, \texttt{register})) \Rightarrow \texttt{viol}_{\perp}$$

At the structural level of an organisation it is often specified which roles are (in)compatible with each other. If two roles are denoted as incompatible this means that these two roles cannot be played by one and the same agent. For example, in the conference management system it might not be allowed for a reviewer to be an author. In our normative multi-agent programming language this can be expressed as:

$$\mathrm{F}(\texttt{rea}(i, \texttt{reviewer}) \wedge \texttt{rea}(i, \texttt{author})) \Rightarrow \texttt{viol}_{\perp}$$

5 Conclusion and Future Work

In this paper we have presented the syntax and operational semantics of a programming language for implementing norm-based multi-agent organisations. These organisations are then developed as a separate entity apart from the agents that will interact with the organisation. In particular, the presented programming language allows for the implementation of a multi-agent organisation by means of norms, being enforced by monitoring, regimenting and sanctioning mechanisms. More specifically, the programming language allows for the expression of more abstract, declarative ought-to-be norms and also allows to refer to concrete actions that have been performed by agents. Although this programming language mainly deals with the normative aspect of an organisation, already some preliminary results were shown of how to deal with the structural aspect of an organisation.

Our ultimate goal is to design a fully-fledged multi-agent programming language based on organisational concepts. The current proposal presented in this paper primarily deals with the normative aspect of multi-agent organisations. Future work aims at extending the programming language with constructs to also support the implementation of the other aspects of multi-agent organisations as mentioned in section 2. In particular, one of our short-term goals is to extend the simple view on roles presented in this paper with a view on roles that better reflects roles as used in multi-agent design methodologies. To what extend norms can be used to ensure well-formedness of the structural specification of the organisation as explored in section 4 should also be further investigated.

Another important issue is that in this paper the architecture is a centralised one. That is to say, the organisation determines the effect agent's actions have on the environment. For the sake of scalability, future research should explore possibilites of decentralising the organisation.

Further, we also aim at incorporating more complex forms of enforcement (e.g., policing agents) and norm types (norms with deadlines, for example). Also the computational cost of the constructs presented should be investigated.

Acknowledgments

This research was supported by the CoCoMAS project funded through the Dutch Organization for Scientific Research (NWO). The authors are grateful for the valuable suggestions, comments, and contributions provided by Davide Grossi.

References

1. Sierra, C., Rodríguez-Aguilar, J., Noriega, P., Esteva, M., Arcos, J.L.: Engineering multi-agent systems as electronic institutions. UPGrade 4 (2004)
2. Zambonelli, F., Jennings, N., Wooldridge, M.: Organisational rules as an abstraction for the analysis and design of multi-agent systems. IJSEKE 11(3), 303–328 (2001)
3. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the GAIA methodology. Acm Tosem 12(3), 317–370 (2003)
4. Dignum, V.: A Model for Organizational Interaction: Based on Agents, Founded in Logic. SIKS Dissertation Series (2003)
5. Hübner, J., Sichman, J., Boissier, O.: Developing organised multi-agent systems using the $\mathcal{M}$oise$^+$ model: Programming issues at the system and agent levels (manuscript)
6. Esteva, M., Rodríguez-Aguilar, J., Sierra, C., Garcia, P., Arcos, J.: On the formal specifications of electronic institutions. In: Agent Mediated Electronic Commerce, The European AgentLink Perspective, pp. 126–147. Springer, London (2001)
7. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: AOSE IV, pp. 214–230 (2004)
8. Esteva, M., Rodríguez-Aguilar, J., Rosell, B., Arcos, J.: Ameli: An agent-based middleware for electronic institutions. In: Kudenko, D., Kazakov, D., Alonso, E. (eds.) AAMAS 2004. LNCS, vol. 3394. Springer, Heidelberg (2005)
9. Watt, D.A.: Programming Language Design Concepts. John Wiley & Sons, Chichester (2004)
10. Dignum, F.: Abstract norms and electronic institutions. In: Proc. of RASTA 2002, Bologna, Italy, pp. 93–104 (2002)
11. Aldewereld, H.: Autonomy vs. Conformity - an Institutional Perspective on Norms and Protocols. PhD Thesis, Universiteit Utrecht (2007)
12. Ricci, A., Viroli, M., Omicini, A.: Give agents their artifacts: the A&A approach for engineering working environments in MAS. In: Proc. of AAMAS 2007, pp. 1–3. ACM Press, New York (2007)
13. Dastani, M., Grossi, D., Tinnemeier, N., Meyer, J.J.: A programming language for normative multi-agent systems (in submission)
14. Meyer, J.J.C., Wieringa, R.J. (eds.): Deontic logic in computer science: normative system specification. John Wiley & Sons, Inc., New York (1994)
15. Grossi, D.: Designing Invisible Handcuffs. Formal Investigations in Institutions and Organizations for MAS. PhD thesis, Utrecht University, SIKS (2007)
16. Searle, J.: The Construction of Social Reality. Free Press (1995)

17. Castelfranchi, C.: Formalizing the informal?: Dynamic social order, bottom-up social control, and spontaneous normative relations. Journal of Applied Logic 1(1-2), 47–92 (2004)
18. Coutinho, L., Sichman, J., Boissier, O.: Modeling organization in mas: A comparison of models. In: Proc. of SEAS 2005, Uberlândia, Brazil (2005)
19. Odell, J., Parunak, H.V.D., Fleischer, M.: The role of roles in designing effective agent organizations. In: Software Engineering for Large-Scale Multi-Agent Systems, Research Issues and Practical Applications, pp. 27–38 (2003)
20. Emerson, E.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, Formal Models and Semantics, Volume B, pp. 995–1072. MIT Press, Cambridge (1990)
21. Plotkin, G.D.: A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus (1981)