Roles and Norms for Programming Agent Organizations

Nick Tinnemeier
Utrecht University
The Netherlands
nick@cs.uu.nl

Mehdi Dastani
Utrecht University
The Netherlands
mehdi@cs.uu.nl

John-Jules Meyer
Utrecht University
The Netherlands
jj@cs.uu.nl

ABSTRACT

We present a programming language for implementing multi-agent systems consisting of a set of individual agents that interact with a computational organization specified in terms of roles, norms and sanctions. We provide an operational semantics of the language that can serve as a basis for the implementation of organizational platforms. The view on roles is motivated by four key properties we deem important for an efficient implementation of agent organizations.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Programming Languages and Software; D.3 [**Programming Languages**]: Miscellaneous

General Terms

Languages, Theory

Keywords

Roles, Norms, Organizations, Operational Semantics

1. INTRODUCTION

The complexity of (distributed) software systems has increased the need for natural, high-level abstractions to aid the developer in the engineering process. Developing software as a multi-agent system is seen as one such suitable abstraction. In particular, the use of organizational abstractions has gained interest within multi-agent systems research [14]. In this approach a multi-agent system is analyzed and designed as a computational organization using social concepts such as roles and norms. Roles allow us to abstract from the individuals that will play them and are particularly useful in the development of open systems in which no assumptions can be made about the internals and behavior of the agents that will interact with it. When little is known about the behavior that the interactants exhibit a mechanism is needed to regulate their behavior. The normative aspect of organizations addresses this issue by effectuating norms the interactants ought to follow. To allow for a straightforward implementation, the concepts used in design should be reflected in a programming language. Not surprisingly also the area of agent programming languages is shifting attention from constructs for implementing single agents to social concepts for programming organizations.

Roles are one of the cornerstones in constructing organizations. Indeed, roles are a recurring concept in agent development methodologies [10] and they are first-class abstractions in several approaches to implementing software systems – within the area of multi-agent systems (cf. [1, 9, 4, 5, 7]) and outside this area (cf. [8, 12]). The role (and the organizational) metaphor accords with how we conceive and structure our world [8]. A host of different interpretations of the concept of role witnesses the lack of consensus on a crisp definition. Kristensen [8] introduces roles as first-class implementation entities in object-oriented programming. In Role-Based Access Control (RBAC) [12] roles are used to flexibly manage the access of users to resources. In [5] and [7] the concept of role is used, comparable to RBAC, as a label to assign norms (such as obligations and permissions) to agents based on the roles they play. Dastani et al. [4] define roles as entities that are specified by the mental attitudes beliefs, goals and planning rules. Agents playing a role internalize these mental attitudes and execute them. Baldoni et al. [1] extend the Java programming language with roles as an integral part of an organization. Roles are implemented as inner classes of the class that implements the organization and empower their players (by providing methods) to access and change the state of the organization (by means of method invocation). Conversely the role requires its players to possess certain capabilities (by requiring certain methods in the class that implements the player). Based on amongst others [1] the JADE (Java Agent DEvelopment) framework is extended to encompass organizations and roles as first-class abstractions in [9].

Despite little agreement on a definition of role still some properties can be identified we think are important in efficiently constructing organizations. Firstly, as argued by Colman and Han [2] for developing adaptable organizations roles need to be *organization-centric* rather than player-centric. This means that a role exists inside an organization as is the case with the approaches of [1] and [9], instead of being an adjunct of its player, an approach taken by [8] and [4]. The organization-centric view promotes the principle of separation of concerns. That is, the organization and its roles can be developed apart from the agents that will play them. Secondly, we observe that roles seem to have a *prescriptive* character. That is to say, they prescribe an expected behavior to their players and in this way guide the players in how to interact with the organization in a mean-

Cite as: Roles and Norms for Programming Agent Organizations, Nick Tinnemeier, Mehdi Dastani, John-Jules Meyer, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 121–128

ingful way. This property is in some way exhibited by all of the previous mentioned approaches and also follows, for instance, from the definition of [10] that introduces a role as "a class that defines a normative behavioral repertoire of an agent." Thirdly, roles are *employable*, meaning that a player can employ its role by delegating it a task without the need to know how this task is executed. The roles of [1, 9, 4, 8] can all be delegated a task. Lastly, we argue that roles should be *declarative*. This accords more with the declarative nature of other concepts such as norms by which organizations are specified. Moreover, most agent-oriented programming languages are based on declarative concepts such as beliefs and goals that are equiped with a well-defined formal semantics. By reclycing these concepts we rely on a great body of research and it allows the programmer to specify roles in terms he/she is already familiar with. Specifically, assumed that roles are employable, attributing goals to roles seems to offer the right level of abstraction, because agents that interact with the organization then need not be concerned with which low-level procedures to use to reach a certain state. The approaches of [4, 5, 12, 7] adopt a declarative view on roles, but only [4] explicitly attributes the same mental attitudes to roles as often used for agents.

In general, we conjecture that all of these properties should be united in an implementation framework for organizations. Moreover, for a clear understanding of the constructs we deem it very important that they are equiped with a verifiable and meaningful semantics. None of the mentioned approaches meets this first point and none of them except for [4] is endowed with a formal semantics. This paper aims to fill this gap by developing an operational semantics for a programming language with the emphasis on organizations and roles that unites all properties as mentioned above. The result serves as a basis for the implementation of organizational platforms. Because roles cannot be understood without the organization they are part of we integrate our work on roles in the already existing research of [3] that defines an operational semantics for an organizational programming language with primary focus on the normative aspect. The existing operational semantics facilitates the integration.

Section 2 introduces the key concepts and their underlying philosophy. Section 3 introduces the syntax of the programming language that is endowed with an operational semantics by section 4. Section 5 concludes this paper. To illustrate our proposal we use an example of a conference management system in which authors can upload papers and that supports reviewers and program chairs with the reviewing process.

2. AGENT ORGANIZATIONS

In our view a multi-agent system is composed of a heterogeneous set of agents that interact with an organization. The property of heterogeneity refers to the agents possibly being implemented by different parties using different (agent) programming languages. What is more, we assume that the developer of the agents is possibly different from that of the organization, i.e. to the organization and its developer agents are black boxes and no assumptions can be made about their behavior. A concrete example of a multi-agent system is depicted by figure 1. The remainder of this section elaborates on all these components.

An organization encapsulates a domain specific state and function, e.g. a database and accompanying operations to

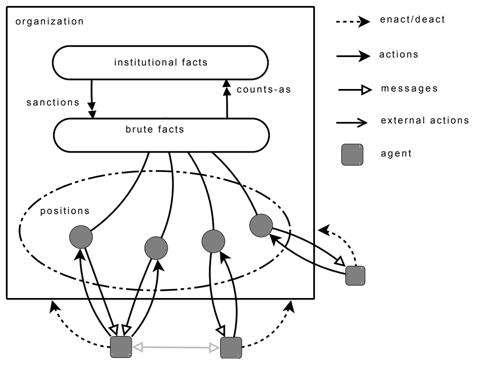

Figure 1: A snapshot of a multi-agent system.

store submitted papers for the conference management system. This domain specific state is described by the *brute facts* and the function is described by external actions that can be used to modify it.

To interact with the organization in a meaningful way agents enact roles that are an integral part of the organization. This accords with organization-centric view. In our framework and similar to that of [4] a role is specified by the same concepts as that of BDI agents, viz. by the mental attitudes belief, goal and plan. The same roles enacted by different agents share a common type of mental state, yet this state may differ in particular details. For example, the reviewer role has goals and beliefs about papers to review, but which papers and how many is specific to the role enactment at hand. When an agent desires to enact a role and the enactment is allowed by the organization, these details are incarnated resulting in an instantiation of the role. Inspired by [10] we refer to the instance of a role by the term *position*: a role assignment that can be occupied by at most one agent at a time.

A player configures its position by inspecting and altering the mental state, in particular, by querying and updating beliefs and goals. A player of the reviewer position can, for example, delegate it a goal to review its papers and update the position's beliefs with the written reviews. Moreover, similar to [4] a position is either active or inactive. When a position is inactive it will not execute any of its plans. This is particularly useful when altering and inspecting the mental state. A position can only be activated and deactivated by its player. The interaction between position and its player is bidirectional. An active position can communicate with its player by sending it messages. The reviewer position can, for example, send its player its assigned papers and inform it about when the reviews should be uploaded. The two way interaction between position and its player implement the properties of employability and prescriptiveness. A player can employ its role by delegating it goals and activating it, whereas the position by sending messages can inform its player about what it is expected to do to achieve a certain goal. A position obtains its computational meaning by the interaction of its player, and therefore differently from [10] in this paper a position cannot exist without an agent associated to it. Indeed, this property explains an important difference between an agent and a position - agents act because they want to, whereas positions act because their players want them to.

In achieving goals for its player a position acts upon the brute state by the performance of external actions. For example, storing a review in a database that can be read by an author position later on. As can be seen in figure 1 only positions can access and alter the brute facts. This promotes the principle of data hiding, i.e. the brute facts are encapsulated by the organization and can only be (partly) read and manipulated by the agents via their positions. Note that positions implement a mechanism comparable to RBAC [12].

Although a position is ideally implemented to obey the organizational rules, we still consider a separate monitoring and sanctioning mechanism crucial for the coordination of the overall system. Why norms and sanctions are needed is explained by the synergy of position and player – the behavior of position and player can only be understood in terms of their interaction – and the fact that disallowing violations of the norms drastically decreases the players' autonomy [6]. Why a separate mechanism is needed is explained by three observations. Firstly, violations of the rules cannot be fully prevented by the position alone. The position cannot by itself, for example, prevent a violation of the rule that reviews should be uploaded before the notification phase starts. Secondly, some rules are hard to implement in the positions because some pertain to the interplay between different position/player pairs, i.e. violations of one or more players might be revealed by actions performed on behalf of another. When the program chair, for example, puts the system in the notification phase this will lead to a violation by all the reviewers which have not yet uploaded their reviews. Thirdly and most importantly, the implementation of norms and sanctions is a different concern than the implementation of roles. Separating their implementation increases the manageability and reusability of both concepts.

Our monitoring and sanctioning mechanism is based on a simple account of counts-as rules as defined by Grossi [6], inspired by the work of Searle [13], and endowed with an operational semantics by Dastani et al. [3]. Counts-as rules normatively assess the brute facts of the organization, for example by specifying that an uploaded paper of more than 15 pages counts as a violation of the page limit. Counts-as rules thus label certain brute states with a normative judgment. This labelling is stored in the *institutional facts*. Sanction rules define the sanctions that should be imposed given a certain normative judgment. Sanction rules are inverted counts-as rules that relate a normative labelling with a set brute facts that should be accommodated. For example, by expressing that a violation of the page limit counts as a fine of 250 euro. Sanctions thus alter the brute facts of the organization. It should be noted that we allow the accommodation of a brute fact to have side effects such as sending a message or issuing a credit card. Besides the judgments that are sanctioned by the sanction rules, [3] introduces a special label $\text{viol}_\perp$ to mean that all external actions that would lead to this special label are made impossible. We acknowledge that there are also other approaches to regulate the behavior of agents, e.g. electronic institutions [5], the deontic component of Moise+ [7] and coordination artifacts [11]. A more elaborate discussion of the normative aspect and a comparison of related work can be found in [3] and is beyond the scope of this paper.

3. SPECIFYING ORGANIZATIONS

This section defines the syntax by which organizations

```
⟨Org⟩       = {"Roles:" {⟨name⟩ ":" ⟨file⟩}
              | "Facts:" {⟨b_literals⟩}
              | "Effects:" ⟨effects⟩
              | "Counts-as rules:" ⟨counts_as⟩
              | "Sanction rules:" ⟨sanctions⟩ };
⟨effects⟩   = {"{"⟨b_literals⟩"}" ⟨atom⟩ "{"⟨b_literals⟩"}"};
⟨counts_as⟩ = {⟨b_literals⟩ "=>" ⟨i_literals⟩};
⟨sanctions⟩ = {⟨i_literals⟩ "=>" ⟨b_literals >⟩};
```

Figure 2: EBNF grammar of an agent organization.

are specified. Henceforth, we use ⟨atom⟩ to denote an atom starting with a lowercase letter. Furthermore, we use two disjoint subsets of the set of ⟨atom⟩s, of which the elements are denoted by ⟨b_atom⟩ and ⟨i_atom⟩ respectively. Given a set of ⟨literal⟩s, positive or negative atoms, we write ⟨b_literal⟩ and ⟨b_literals⟩ to denote respectively the literal and set of literals containing solely ⟨b_atom⟩s. The tokens ⟨i_literal⟩ and ⟨i_literals⟩ are defined in a similar fashion. The syntax by which organizations are specified is depicted in figure 2.

Note that the specification of an organization does not include a specification of the agents, since these are situated outside the organization. The roles of the organization are attributed a ⟨name⟩ and are specified in a ⟨file⟩. The (brute) facts specify the initial state. The conference management system, for instance, is initially closed denoted by a fact *phase*(*closed*) and the list of uploaded papers is initially empty denoted by $papers([])$. The brute facts change during the execution of the organization due to external actions of the positions. The effect of these external actions on the brute state is specified by means of the effect rules, triples consisting of an action name enclosed by two sets of brute facts. The antecedent specifies the state of the brute facts in which the action can be performed and the consequent specifies the effect that should be accommodated in the brute facts. As an example consider the following effect rule that specifies that during the submission phase an author A can upload a paper P which is then stored in the list of received papers:

```
{ phase(submission), papers(Rs) }
  upload(A,P)
{ not papers(Rs), papers([(A,P)|Rs])}
```

The counts-as rules associate a normative judgment with a certain brute state and the sanction rules specify the sanctions in terms of the brute facts that should be accomodated given such a judgment. For example, consider a counts-as rule that labels the fact that an author A sent a paper P of more than 15 pages as a violation:

```
{papers(As),(A,P)∈As,pages(P)>15} ⇒ {viol(s,A)}
```

which is sanctioned by a fine of 250 euro as specified by the following sanction rule:

```
{viol(s,A)} ⇒ {fined(A,250)}
```

For the specification of the syntax of roles the atom with predicate name '`construct`' is denoted by ⟨constr_atom⟩ and the atom with predicate '`destruct`' by ⟨destr_atom⟩. The syntax for specifying roles is defined as depicted in figure 3.

For the sake of generality we leave the language by which roles are implemented largely unspecified. We encourage the reuse of (parts of) existing BDI-based programming language endowed with a formal semantics. Without loss of generality we assume that a role is specified by a set of initial beliefs denoting the information about its environment and actions to modify it, a set of goals denoting the desired

```
⟨Role⟩       = "Beliefs:" ⟨belief⟩+
               "Goals:" ⟨goal⟩+
               "Constructors:" ⟨constr⟩+
               "PG-rules:" ⟨pgrule⟩+
               "Destructors:" ⟨destr⟩+;
⟨constr⟩     = ⟨constr_atom⟩ "<-" ⟨test⟩ "|" ⟨plan⟩;
⟨pgrules⟩    = ([⟨goalquery⟩] "<-" ⟨belquery⟩ "|" ⟨plan⟩)+;
⟨destr⟩      = ⟨destr_atom⟩ "<-" ⟨test⟩ "|" ⟨plan⟩;
⟨test⟩       = "B(" ⟨belquery⟩ ")" | "G(" ⟨goalquery⟩ ")"
             | "E(" ⟨envquery⟩ ")" | "I(" ⟨instquery⟩ ")"
             | ⟨test⟩ "&" ⟨test⟩;
⟨envquery⟩   = "true" | ⟨b_literal⟩ | ⟨envquery⟩ "and" ⟨envquery⟩
             | ⟨envquery⟩ "or" ⟨envquery⟩ | "(" ⟨envquery⟩ ")";
⟨instquery⟩  = "true" | ⟨i_literal⟩ | ⟨instquery⟩ "and" ⟨instquery⟩
             | ⟨instquery⟩ "or" ⟨instquery⟩ | "(" ⟨instquery⟩ ")";
⟨belquery⟩   = "true" | ⟨literal⟩ | ⟨belquery⟩ "and" ⟨belquery⟩
             | ⟨belquery⟩ "or" ⟨belquery⟩ | "(" ⟨belquery⟩ ")";
⟨goalquery⟩  = "true" | ⟨atom⟩ | ⟨goalquery⟩ "and" ⟨goalquery⟩ |
             | ⟨goalquery⟩ "or" ⟨goalquery⟩ | "(" ⟨goalquery⟩ ")";
```

Figure 3: EBNF grammar of a role.

state of the world, external actions to act upon the brute facts and actions to communicate with its player. Inspired by object-oriented programming we introduce the notion of a constructor, a plan that is executed upon instantiation with the responsibility to put a freshly created position in a valid state. A position is instantiated when an agent enacts a role. In enacting a role the enactant provides a list of arguments with which it wishes to enact the role. These arguments are matched with the parameters of the constructor. Because a role can only be enacted under certain circumstances it is guarded by a pre-condition, a query evaluated over the beliefs and goals (marked by B and G) of the position, and the institutional (marked by I) and brute (marked by E) state. For example, a constructor:

```
construct(name(N),amount(Nr))
  <- E(phase(reg) and invited(N)) & B(Nr≥3) | π
```

specifies that the reviewer role can only be enacted if enactant N is invited and agrees to minimally review three papers. When these requirements are met the plan π is executed that stores the name of the reviewer as a belief $name(N)$, adopts a goal $review(Nr)$ and registers the reviewer to the system by performing an external action. A role is endowed with a set of planning goal rules that describe which plan should be used to achieve a goal given certain beliefs. Consider, for example, a pg-rule of the reviewer role for the goal $review(Nr)$:

```
review(Nr)
  <- E(phase(rev)) & B(not notified(rev)) | π
```

that in the reviewing phase and has not notified its player yet, notifies it about its assigned papers and requests it to review them. The destructor of a role is a plan that is called upon de-enactment of a position. Similar to enactment de-enactment can only take place under certain circumstances and therefore we equip the destructor with a guard. For example, a reviewer can only de-enact its role in the notification phase when it has uploaded all its reviews and which will de-register the reviewer:

```
destruct()
  <- E(phase(not)) & B(nr(X) and up(X)) | π
```

4. EXECUTING ORGANIZATIONS

This section defines the semantics of organizations as specified by the syntax as defined in section 3. The semantics is defined by means of a transition system that is built of a set of axioms and transition rules for deriving transitions that describe how the configuration of a program transforms from one into another.

4.1 Preliminaries

In our system roles are a core ingredient in implementing organizations. As an instance of a role a position has a state that changes during execution.

Definition 1. A position, typically denoted by ρ, is a tuple $\langle p, \Sigma, \Gamma, \Pi, f\rangle$, in which:

- p is the label uniquely identifying the position. This name has the form $i.k$ with k a unique identifier and i an agent identifier denoting the player. From here on we say that a position p is played by i iff $p = i.k$;
- Σ is a set of belief expressions ⟨belief⟩ representing the belief base of the position;
- Γ is a set of goal expressions ⟨goal⟩ representing the goal base of the position;
- Π is a set of adopted plans generated by the pg-rules;
- $f \in \{\top, \bot\}$ is a boolean flag indicating the activation status of the position, $\top$ denotes active and $\bot$ inactive.

In the following we say that a position $\langle p, \Sigma, \Gamma, \Pi, f\rangle$ with $f = \top$ is active. □

Besides the positions that are enacted by agents an organization is characterized by brute facts and institutional facts. Agents interact with the organization by performing special designated actions such as enacting a role, activating and altering a position. These actions can be perceived by the organization and are stored in its event base.

Definition 2. An organization, typically denoted by O, is defined as a tuple $\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle$, in which:

- P is a set of positions (role instances);
- B is a set of ⟨b_atom⟩ expressions describing the brute state of the organization;
- I is a set of ⟨i_atom⟩ expressions describing the institutional state of the organization;
- E is a list of ⟨atom⟩s, the event base that stores the actions that are perceived by the organization. We write $e.\mathrm{E}$ to denote a list with head e and tail E and we write $\mathrm{E}.e$ to mean that element e is appended at the end of the list. □

The execution of an organization cannot be understood apart from its context, that is the multi-agent system it is part of. The organization changes due to the actions that are performed by agents and the execution of positions.

Definition 3. A multi-agent system is a tuple $\langle \mathrm{A}, \mathrm{O}\rangle$ with: A a set of agents interacting with organization O. □

The condition of a constructor and destructor of a role allow a test on its intrinsic state (goals and beliefs) and extrinsic state (brute and institutional facts). Hence the definition of the special entailment operator $\models_t$ that evaluates such a test.

Definition 4. Given ⟨test⟩ expressions $\varphi(\overline{x})$ and $\varphi'(\overline{y})$ in which sets of free variables $\overline{x}$ and $\overline{y}$ occur, ⟨belquery⟩ expression ϕ_b, ⟨goalquery⟩ expression ϕ_g, ⟨envquery⟩ expression ϕ_e, ⟨instquery⟩ expression ϕ_i, first-order entailment relation

$\models$ and substitution functions τ and τ', the entailment relation $\models_t$ that evaluates test expressions w.r.t. beliefs, goals, brute facts and institutional facts $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I})$ is defined as:

- $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models_t \mathtt{B}(\phi_b)\tau$ iff $\Sigma \models \phi_b\tau$
- $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models_t \mathtt{G}(\phi_g)\tau$ iff $\exists \gamma \in \Gamma$ s.t. $\gamma \models \phi_g\tau$
- $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models_t \mathtt{E}(\phi_e)\tau$ iff $\mathrm{B} \models \phi_e\tau$
- $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models_t \mathtt{I}(\phi_i)\tau$ iff $\mathrm{I} \models \phi_i\tau$
- $(\Sigma, \Gamma, B, I) \models (\varphi(\overline{x})\&\varphi'(\overline{y}))\tau$ iff $\exists\tau_1 : [\tau_1 = \tau|\overline{x}$ and $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models \varphi\tau_1$ and $\exists\tau_2 : [\tau_2 = \tau|(\overline{y} \setminus \overline{x})$ and $(\Sigma, \Gamma, \mathrm{B}, \mathrm{I}) \models \varphi'\tau_1\tau_2]]$

with '|' to be read as 'restricted to the domain'.

The brute state of the organization is normatively assessed by applying the counts-as rules and sanctions are imposed by applying the sanction rules. This boils down to adding the consequences of the (normative or sanctioning) rules of which the antecedent is satisfied to the (brute or institutional) facts. To this end we define the closure of a set of facts under a set of rules.

Definition 5. Let $l = \Phi(\overline{x}) \Rightarrow \Psi(\overline{y})$ be a rule with Φ and Ψ sets of first-order literals in which sets of variables $\overline{x}$ and $\overline{y}$ occur such that $\overline{y} \subseteq \overline{x}$ and all variables are universally quantified in the widest scope. Moreover, we respectively denote the condition and consequent of a rule l by $cond_l$ and $cons_l$. Given a set R of rules and a set X of ground literals the set of applicable rules in X is defined as:

$$Appl^R(X) = \{l\tau : l \in R \wedge X \models cond_l\tau \text{ for a substitution } \tau\}$$

Moreover, let the closure of a set of facts X under a set of rules R be inductively defined as:

$$\begin{array}{lcl} \mathbf{B} : Cl_0^R(X) & = & X \cup (\bigcup_{l \in Appl^R(X)} cons_l) \\ \mathbf{S} : Cl_{n+1}^R(X) & = & Cl_n^R(X) \cup (\bigcup_{l \in Appl^R(Cl_n^R(X))} cons_l) \end{array}$$

then the ground closure of X under R is defined as $Cl^R(X)$ for the minimal k such that $Cl_{k+1}^R(X) = Cl_k^R(X)$. $\square$

To define the semantics of a multi-agent system that consists of agents and an organization we use four types of transitions, namely transitions that concern the execution of: the multi-agent system, individual agents, organizations and the positions of the organization. Each of these transitions is defined by its own transition system.

4.2 Execution of Agents and Positions

The basic types are the transitions that define the execution of an agent and the transitions that define the execution of a position, both denoted by an arrow $\rightarrow$. The transition systems that define these transitions are out of scope of this paper. The agent transitions because agents are treated as black boxes and the position transitions to preserve generality of our approach. In fact, the semantics of a position is similar to that of agent programming languages and is defined elsewhere. Without loss of generality we assume that agents can perform observable actions α (of type $\langle$atom$\rangle$) to interact with the organization and enacted positions, and can receive messages m (also of type $\langle$atom$\rangle$) that are sent by the organization. Moreover, we assume agents can execute non-observable actions that only affect their internal state. More precisely, we presume transition rules by which the following transitions can be derived for an agent configuration a identified by i:

1. $a \xrightarrow{\alpha!} a'$: performance of observable action α
2. $a \xrightarrow{m_{i,j}?} a'$: reception of message m sent by j
3. $a \longrightarrow a'$: performance of an internal action

Similarly, we assume that active positions can execute external actions δ (of type $\langle$atom$\rangle$) that at the organizational level change the brute state. We also assume that positions can make internal transitions that only affect their internal state. Examples of such internal actions are updates of the belief base and the addition and deletion of goals. Moreover, we assume that positions can send messages to their players. More precisely, we assume transition rules by which the following transitions for an active position ρ can be derived:

1. $\rho \xrightarrow{\delta!} \rho'$: performance of external action δ
2. $\rho \xrightarrow{m!} \rho'$: sending of message m to its player
3. $\rho \longrightarrow \rho'$: performance of an internal action

Note that these transitions do not define how the organization receives and responds to these actions as performed by agents and positions. The effect of external actions performed by positions is determined by the organization. Consequently, these transition rules are defined in the organization transition system. Also the response of the organization to the agent's observable actions is defined in the organization transition system, whereas the reception of these actions is defined by the multi-agent transition system. In defining these compositional transition systems we use a bottom-up approach. That is to say, the transition system from which the transitions of the organization derive is defined first and these transitions will be denoted by an arrow $\mapsto$. The transitions of the multi-agent system are denoted by an arrow $\Rightarrow$ and are defined in terms of the transitions of the agents and the organization.

4.3 Execution at the Organizational Level

In defining the rules for organizations we assume an organization configuration $\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle$. To start with, we define the rule that specifies how an organization responds to the reception of an observable action performed by an agent.

Rule 1. Let α be an external action. Then the rule for perceiving external actions is defined as:

$$\frac{}{\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle \overset{\alpha?}{\longmapsto} \langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}.\alpha\rangle}$$

An organization can thus alway perceive the agents' actions and stores them at the end of the list of perceived actions. The next transitions specify how the organization responds to various actions such as enactment and deenactment of roles and actions by which players configure their positions.

Before we explain how agents take up roles we first introduce the transition rules that define the execution of positions, because these transitions will be used in defining position enactment and de-enactment later on. Most actions of a position only affect its internal state and leave the state of the organization unchanged. Recall that a position can only advance in its computation if it is activated by its player. Internal transitions are defined by the following rule.

Rule 2. Let $\rho = \langle p, \Sigma, \Gamma, \Pi, \top\rangle$ be an active position, then the rule for internal actions is defined as:

$$\frac{\rho \in \mathrm{P} \quad \rho' = \langle p, \Sigma', \Gamma', \Pi', \top\rangle \quad \rho \rightarrow \rho'}{\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle \mapsto \langle (\mathrm{P} \setminus \{\rho\}) \cup \{\rho'\}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle}$$

The brute state of the organization can only be altered by the positions by means of the performance of external actions. The organization uses the effect rules to determine the effect of these actions. Once the effect of an external action is effectuated the new brute state is assessed by means of the counts-as rules and sanctions are imposed accordingly by means of the sanction rules. An external action can only be performed when it does not lead to a $\text{viol}_\perp$. This way regimentation is implemented. The next rule defines the performance of external actions by a position.

Rule 3. Let C be the set of counts-as rules, S be the set of sanction rules, ρ be an active position and δ an external action. Further, let $\text{up}(\mathrm{B},\delta)$ be a function that by means of the effect rules determines the effect on the brute facts B of performing action δ. Then the rule for external actions is defined as:

$$\frac{\rho\in\mathrm{P}\quad \rho\xrightarrow{\delta!}\rho'\quad \mathrm{B}'=\text{up}(\mathrm{B},\delta)\quad \mathrm{I}'=Cl^C(\mathrm{B}')\setminus(\mathrm{B}')\qquad \mathrm{I}'\not\models\text{viol}_\perp\quad \mathrm{B}''=Cl^S(\mathrm{I}')\setminus(\mathrm{I}')}{\langle\mathrm{P},\mathrm{B},\mathrm{I},\mathrm{E}\rangle\mapsto\langle(\mathrm{P}\setminus\{\rho\})\cup\{\rho'\},\mathrm{B}'\cup\mathrm{B}'',\mathrm{I}',\mathrm{E}\rangle}$$

An agent i indicates that it desires to take up a role r with a list of parameters Φ by performing an action $enact(i,r,\Phi)$. The next transition rule defines role enactment. A role can be enacted only if the organizational rules are respected. More precisely, when the arguments Φ match with the arguments of some constructor of which the pre-condition is satisfied. If so, a position ρ is instantiated out of the role specification by executing its constructor. The execution of the constructor boils down to the execution of ρ with the constructor's plan in its plan base to a position ρ' with an empty plan base. This execution is denoted by a sequence of transitions: $\langle\{\rho\},\mathrm{B},\mathrm{I},\mathrm{E}\rangle\mapsto^*\langle\{\rho'\},\mathrm{B}',\mathrm{I}',\mathrm{E}\rangle$ that are to be derived from transition rules 2 and 3. Note that the execution of the constructor might both change the internal state of freshly instantiated position ρ as well as the brute and normative state of the organization. Recall that according to these rules only active positions can be executed, whereas similar to [4] an enacted position is initially inactive. This explains the need for de-activating active position ρ' (denoted by ρ''). If the position is succesfully instantiated the agent is informed about this and is sent the identifier of the position such that it can interact with it.

Rule 4. Let $\texttt{construct}(\Psi)\leftarrow\varphi\mid\pi$ be a constructor of r with Φ a list of $\langle$groundatom$\rangle$s and let Ψ be a list of $\langle$atom$\rangle$s. Moreover, assume a function $\text{unify}(\Psi,\Phi)$ that evaluates to the most general unifier τ of Ψ with Φ if they are unifiable and returns $\perp$ otherwise. Then the rule for role enactment is defined as:

$$\frac{\text{unify}(\Phi,\Psi)=\tau_1\quad(\Sigma,\Gamma,\mathrm{B},\mathrm{I})\models_t\varphi\tau_1\tau_2\qquad \langle\{\rho\},\mathrm{B},\mathrm{I},\mathrm{E}\rangle\mapsto^*\langle\{\rho'\},\mathrm{B}',\mathrm{I}',\mathrm{E}\rangle}{\langle\mathrm{P},\mathrm{B},\mathrm{I},enact(i,r,\Phi).\mathrm{E}\rangle\overset{m_{i,p}!}{\longmapsto}\langle\mathrm{P}\cup\{\rho''\},\mathrm{B}',\mathrm{I}',\mathrm{E}\rangle}$$

where $p\ = i.k$ in which k is a unique identifier
and $\rho\ = \langle p,\Sigma,\Gamma,\{\pi\tau_1\tau_2\},\top\rangle$
and $\rho'\ = \langle p,\Sigma',\Gamma',\{\},\top\rangle$
and $\rho''\ = \langle p,\Sigma',\Gamma',\{\},\perp\rangle$
and $m\ = msg(enacted(r))$

Enactment of a role might not be successful for various reasons. For example, when the pre-conditions of role enactment are not respected or in case the parameters do not match with any constructor. The definition of transition rules that pertain to unsuccesful enactment is out of the scope of this paper.

A position can only be de-enacted if the condition of some of its destructors is satisfied. A request of an agent i to de-enact position p is done by the performance of an action $deact(p)$. Note that the agent's identity is coded in the identifier p, so an agent can only interact with its own position(s). If the de-enactment of the position is allowed, it is de-instantiated by executing its destructor.

Rule 5. Let ρ be a position played by agent i and let $\texttt{destruct}()\leftarrow\varphi\mid\pi$ be a destructor of ρ. Then the rule for role de-enactment is defined as:

$$\frac{\rho=\langle p,\Sigma,\Gamma,\Pi,\perp\rangle\quad\rho\in\mathrm{P}\qquad(\Sigma,\Gamma,\mathrm{B},\mathrm{I})\models_t\varphi\tau\quad\langle\{\rho'\},\mathrm{B},\mathrm{I},\mathrm{E}\rangle\mapsto^*\langle\{\rho''\},\mathrm{B}',\mathrm{I}',\mathrm{E}\rangle}{\langle\mathrm{P},\mathrm{B},\mathrm{I},deact(p).\mathrm{E}\rangle\overset{m_{i,p}!}{\longmapsto}\langle\mathrm{P}\setminus\{\rho\},\mathrm{B}',\mathrm{I}',\mathrm{E}\rangle}$$

where $\rho'\ = \langle p,\Sigma,\Gamma,\{\pi\tau\},\top\rangle$
and $\rho''\ = \langle p,\Sigma',\Gamma',\{\},\top\rangle$
and $m\ = msg(deacted(p))$

An agent changes the operation status of a position p it plays by means of an $activate(p)$ and $deactivate(p)$ action, respectively. A position can only be activated if it is inactive and can only be deactivated if it is active. The following two transitions define the organization's response to an *activate* and a *deactivate*.

Rule 6. Let $\rho=\langle p,\Sigma,\Gamma,\Pi,\perp\rangle$ be a position. The rule for activating a position is defined as:

$$\frac{\rho\in\mathrm{P}}{\langle\mathrm{P},\mathrm{B},\mathrm{I},activate(p).\mathrm{E}\rangle\mapsto\langle\mathrm{P}',\mathrm{B},\mathrm{I},\mathrm{E}\rangle}$$

where $\mathrm{P}'=(\mathrm{P}\setminus\{\rho\})\cup\{\langle p,\Sigma,\Gamma,\Pi,\top\rangle\}$

Rule 7. Let $\rho=\langle p,\Sigma,\Gamma,\Pi,\top\rangle$ be an active position. The rule for deactivating a position is defined as:

$$\frac{\rho\in\mathrm{P}}{\langle\mathrm{P},\mathrm{B},\mathrm{I},deactivate(p).\mathrm{E}\rangle\mapsto\langle\mathrm{P}',\mathrm{B},\mathrm{I},\mathrm{E}\rangle}$$

where $\mathrm{P}'=(\mathrm{P}\setminus\{\rho\})\cup\{\langle p,\Sigma,\Gamma,\Pi,\perp\rangle\}$

An agent can inspect and alter the mental state of a position it has enacted. In particular, an agent can inspect and update its beliefs and goals. The next transition rules define the inspection and modification of positions. The performance of action $test(p,\phi)$ indicates that the player of p performs test ϕ on position p. The organization responds to such an action by sending the agent a message with the resulting substitution τ of the test, if ϕ is derivable. Special substitution ϵ is used to indicate that ϕ is not derivable from the position's mental state. Note that a player can only perform a test on a position's mental state and not on the brute and institutional facts.

Rule 8. Let $\rho=\langle p,\Sigma,\Gamma,\Pi,f\rangle$ be a position played by agent i and let ϕ be a $\langle$test$\rangle$ expression containing solely $\langle$belquery$\rangle$ and $\langle$goalquery$\rangle$ expressions. The rule for performing an internal test on a position is defined as:

$$\frac{\rho\in\mathrm{P}}{\langle\mathrm{P},\mathrm{B},\mathrm{I},test(p,\phi).\mathrm{E}\rangle\overset{m_{i,p}!}{\longmapsto}\langle\mathrm{P},\mathrm{B},\mathrm{I},\mathrm{E}\rangle}$$

$$\text{where } m=\begin{cases}msg(test(\phi),\tau) & \text{if } (\Sigma,\Gamma,\mathrm{B},\mathrm{I})\models_t\phi\tau\\ msg(test(\phi),\epsilon) & \text{otherwise}\end{cases}$$

A goal ϕ can be delegated to a position p by its player by means of an action $adoptg(p,\phi)$. Dropping a goal proceeds similarly by action $adoptg(p,\phi)$. Goals can always be delegated to and dropped from a position by its enacting agent. This is specified by the following rule.

Rule 9. Let $\rho = \langle p, \Sigma, \Gamma, \Pi, f\rangle$ be a position and let $\alpha = adoptg/dropg(p, \phi)$ with ϕ a $\langle$goal$\rangle$ expression. The rule for adopting/dropping a goal to/from a position is defined as:

$$\frac{\rho \in \mathrm{P}}{\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \alpha.\mathrm{E}\rangle \mapsto \langle (\mathrm{P} \setminus \{\rho\}) \cup \rho', \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle}$$

$$\text{where } \rho' = \begin{cases} \langle p, \Sigma, \Gamma \cup \{\phi_g\}, \Pi, f\rangle & \text{if } \alpha = adoptg(p, \phi) \\ \langle p, \Sigma, \Gamma \setminus \{\phi_g\}, \Pi, f\rangle & \text{if } \alpha = dropg(p, \phi) \end{cases}$$

An agent i can also alter the belief base of a position p by adding or removing a belief ϕ with actions $adoptb(i, p, \phi)$ and $dropb(i, p, \phi)$, respectively. The following transition rule defines the addition and deletion of beliefs to a position.

Rule 10. Let $\rho = \langle p, \Sigma, \Gamma, \Pi, f\rangle$ be a position played by agent i and let $\alpha = adoptb/dropb(i, p, \phi)$ in which ϕ is a $\langle$groundatom$\rangle$. Moreover, let $\oplus$ be a consistency preserving update operator. The rule for adopting/dropping a belief to/from a position is defined as:

$$\frac{\rho \in \mathrm{P}}{\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \alpha.\mathrm{E}\rangle \mapsto \langle (\mathrm{P} \setminus \{\rho\}) \cup \rho', \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle}$$

$$\text{where } \rho' = \begin{cases} \langle p, \Sigma \oplus \{\phi\}, \Gamma, \Pi, f\rangle & \text{if } \alpha = adoptb(i, p, \phi) \\ \langle p, \Sigma \setminus \{\phi\}, \Gamma, \Pi, f\rangle & \text{if } \alpha = dropb(i, p, \phi) \end{cases}$$

Not only an agent communicates with its position by performing actions, but also the position can send messages to its player, i.e. as defined by the position transition 2 of the previous section. Whenever a position sends a message the organization propagates this message to the multi-agent level and assures that this message will always be sent to its player. This is expressed by the following rule.

Rule 11. Let ρ be a position identified by p and played by agent i, and let m be a message. Then the rule for sending messages is defined as:

$$\frac{\rho \in \mathrm{P} \quad \rho \xrightarrow{m!} \rho'}{\langle \mathrm{P}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle \overset{m_{i,p}!}{\longmapsto} \langle (\mathrm{P} \setminus \{\rho\}) \cup \rho', \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle}$$

4.4 Execution at the multi-agent system level

Hitherto we defined that agents can perform actions to interact with the organization, can receive messages sent by the organization and their positions, and we defined how the organization responds to actions performed by agents. Having defined these rules we are now able to define the transition system that brings all this together and specifies the execution of a multi-agent system. In defining these rules we assume a multi-agent system $\langle \mathrm{A}, \mathrm{O}\rangle$. We start with the rule that defines the communication between organization and agent. Recall that the organization sends messages in transitions derived by rules 4 (result of enact), 5 (result of deact), 8 (result of test) and 11 (message sent by position). Note that due to the construction of rule 11 agents will only receive messages from positions they play.

Rule 12. Let m be a message and a be an agent identified by i. Then the rule for communication from organization to agent is defined as:

$$\frac{\mathrm{O} \overset{m_{i,p}!}{\longmapsto} \mathrm{O}' \quad a \xrightarrow{m_{i,p}?} a'}{\langle \mathrm{A}, \mathrm{O}\rangle \Rightarrow \langle (\mathrm{A} \setminus \{a\}) \cup \{a'\}, \mathrm{O}'\rangle}$$

The following rule defines the performance of an observable action by the agent that is observed by the organization. Recall that the perception of actions by the organization is defined by transition rule 1.

Rule 13. Let α be an observable action. Then the rule for the agent interacting with the organization is defined as:

$$\frac{a \xrightarrow{\alpha!} a' \quad \mathrm{O} \overset{\alpha?}{\longmapsto} \mathrm{O}'}{\langle \mathrm{A}, \mathrm{O}\rangle \Rightarrow \langle (\mathrm{A} \setminus \{a\}) \cup \{a'\}, \mathrm{O}'\rangle}$$

The above rules define interaction between organization and agents that, consequently, change the state of these two components. Both organization and agent can make internal transitions that only affect their own state. The last two transition rules define the case in which agent and organization make an internal transition. Internal transitions of the organization can be derived by transition rules 2 (internal execution of position) and 3 (external action executed by position).

Rule 14. The rule for an internal transition of an agent at the multi-agent level is defined as:

$$\frac{a \to a'}{\langle \mathrm{A}, \mathrm{O}\rangle \Rightarrow \langle (\mathrm{A} \setminus \{a\}) \cup \{a'\}, \mathrm{O}\rangle}$$

Rule 15. The rule for an internal transition of an organization at the multi-agent level is defined as:

$$\frac{\mathrm{O} \mapsto \mathrm{O}'}{\langle \mathrm{A}, \mathrm{O}\rangle \Rightarrow \langle \mathrm{A}, \mathrm{O}'\rangle}$$

4.5 Exemplar multi-agent system execution

An execution of a multi-agent system is a sequence of configurations that can be generated by applying transition rules to an initial configuration and thus shows a possible behavior of the system. We conclude with an example execution involving the enactment of a role by an agent and for which we assume the following initial multi-agent system configuration:

$$\langle \{a_j, ...\}, \langle \emptyset, \mathrm{B}, \mathrm{I}, enact(j, r, name(john), amount(3)).\mathrm{E}'\rangle\rangle$$

in which we have an agent named 'john' that is identified by j and has just performed an action to enact the reviewer role (abbreviated by r) with the agreement to review three papers. Note that currently no position is enacted. Next, we show how the organization handles the request to enact the reviewer role. The transitions that can be derived at the multi-agent system level are composed of transitions made by the organization, agents and positions. We show the complete derivation of the transition that explains the enactment of the reviewer role at the multi-agent system level in a bottom-up fashion.

Recall the constructor of this role (as specified in section 3) that consecutively stores the name of the reviewer as a belief $name(N)$, adopts a goal $review(Nr)$ and registers the reviewer to the system by performing an external action. Recall the transitions of a position as presented in section 4.2. Assumed that this role has no initial beliefs and goals, the execution of its constructor is explained by the following example sequence of position transitions in which δ denotes the action of registering 'john' to the system:

$$\begin{array}{rl} \langle j.1, \{\}, \{\}, \{\pi\}, \top\rangle & \to \\ \langle j.1, \{name(john)\}, \{\}, \{\pi'\}, \top\rangle & \to \\ \langle j.1, \{name(john)\}, \{review(3)\}, \{\pi''\}, \top\rangle & \xrightarrow{\delta!} \\ \langle j.1, \{name(john)\}, \{review(3)\}, \{\}, \top\rangle & \end{array}$$

The first two transitions do not change the organization, but the last one does; the registration changes the brute state to contain a fact $registered(john)$ as specified by some effect rule. Assumed that registering is not debt to some counts-as rule, given the three above transitions we apply rules 2 (for the first two transitions) and 3 (for the third transition) to derive the following transitions that view the execution of the constructor in the context of the organization:

$$\begin{array}{rl}
\langle\{\langle j.1, \{\}, \{\}, \{\pi\}, \top\rangle\}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle & \mapsto \\
\langle\{\langle j.1, \{name(john)\}, \{\}, \{\pi'\}, \top\rangle\}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle & \mapsto \\
\langle\{\langle j.1, \{name(john)\}, \{review(3)\}, \{\pi''\}, \top\rangle\}, \mathrm{B}, \mathrm{I}, \mathrm{E}\rangle & \mapsto \\
\langle\{\langle j.1, \{name(john)\}, \{review(3)\}, \{\}, \top\rangle\}, \mathrm{B}', \mathrm{I}, \mathrm{E}\rangle &
\end{array}$$

Note that we do not show the brute state. The constructor of the reviewer role can only be executed if its pre-condition is satisfied. Recall that for this role this means that 'john' should be invited as a reviewer and agrees to minimally review three papers. From the parameters of the enactment we already know that 'john' wants to review three papers. Lets assume that 'john' is invited, i.e. $B \models invited(john)$ then by applying rule 4 for the previous sequence of transitions we obtain the following transition that pertains to the handling of the enact action at the level of the organization:

$$\begin{array}{rl}
\langle\emptyset, \mathrm{B}, \mathrm{I}, enact(j, r, name(john), amount(3)).\mathrm{E}'\rangle & \overset{m_{j,j.1}!}{\longmapsto} \\
\langle\{\langle j.1, \{name(john)\}, \{review(3)\}, \{\}, \bot\rangle\}, \mathrm{B}', \mathrm{I}, \mathrm{E}'\rangle &
\end{array}$$

The reviewer role is thus successfully instantiated and an inactive position played by 'john' is added to the organization. What is more, the organization sends 'john' a message $msg(enacted(r))$ (abbreviated by m) to notify him about the enactment. Out of this transition and our earlier assumption that an agent can always receive messages sent by the organization, i.e. $a_j \xrightarrow{m_{j,j.1}?} a'_j$ we can derive the final transition that explains the handling of the enactment at the multi-agent system level by applying rule 12:

$$\begin{array}{l}
\langle\{a_j, ...\}, \langle\emptyset, \mathrm{B}, \mathrm{I}, enact(j, r, name(john), amount(3)).\mathrm{E}'\rangle\rangle \Rightarrow \\
\langle\{a'_j, ...\}, \langle\{\langle j.1, \{name(john)\}, \{review(3)\}, \{\}, \bot\rangle\}, \mathrm{B}', \mathrm{I}, \mathrm{E}'\rangle\rangle
\end{array}$$

Now interaction between the reviewer position and 'john' can start. Once activated, the position will send him the assigned papers and guide him through the rest of the reviewing process. Also, 'john' can interact with the position by, for example, uploading its reviews once finished.

5. CONCLUSION

In this paper we have presented a programming language for implementing open multi-agent systems as consisting of a set of individual agents that interact with an organization specified in terms of roles, norms and sanctions. Our view on roles has been motivated by four key properties, in particular we see roles as being organization-centric, prescriptive, employable and declarative. We have endowed this language with an operational semantics that can serve as a basis for the implementation of open multi-agent system platforms.

To fully exploit the potential of open multi-agent systems for developing complex distributed software we see the following directions for future research. In this paper we did not address how the agents obtain knowledge about which roles can be enacted and which goals can be delegated to them. This issue is left for future research. We think the solution of Yellow Pages as used in [9] is a promising first step in this direction. Moreover, in [14] the importance of rules to the organizational structure has been emphasized. To us it seems that there is a resemblance between our notion of norms and the formalism of [14]. We see norms as a potential candidate in expressing relationships and constraints between roles. To end with, we stress the importance of a proof of concept in the form of an implementation. We have already commenced with an implementation of the framework as proposed in this paper.

6. REFERENCES

[1] M. Baldoni, G. Boella, and L. van der Torre. Roles as a coordination construct: Introducing powerjava. In *Proc. of 1st Int. Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, 2005.

[2] A. Colman and J. Han. Roles, players and adaptable organizations. *Applied Ontology*, 2(2):105–126, 2007.

[3] M. Dastani, N. A. M. Tinnemeier, and J.-J. C. Meyer. A programming language for normative multi-agent systems. In V. Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter 16. IGI Global, 2008.

[4] M. Dastani, M. van Riemsdijk, J. Hulstijn, F.Dignum, and J.-J. C. Meyer. Enacting and deacting roles in agent programming. In *Proc. of the 5th Int. Workshop on AOSE*, page 3382, 2004.

[5] M. Esteva, J. Rodríguez-Aguilar, B. Rosell, and J. Arcos. Ameli: An agent-based middleware for electronic institutions. In *Proc. of AAMAS 2004*, New York, US, 2004.

[6] D. Grossi. *Designing Invisible Handcuffs. Formal Investigations in Institutions and Organizations for MAS*. PhD thesis, Utrecht University, SIKS, 2007.

[7] J. F. Hubner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *Int. J. Agent-Oriented Softw. Eng.*, 1(3/4):370–395, 2007.

[8] B. Kristensen. Object-oriented modeling with roles. In *Proc. of the 2nd Int. Conf. on OO Information Systems*, pages 57–71. Springer-Verlag, 1995.

[9] R. G. Matteo Baldoni, Valerio Genovese and L. van der Torre. Adding organizations and roles as primitives to the jade framework. In *Proc. of the 3rd Int. Workshop on Normative MAS*, 2008.

[10] J. J. Odell, H. Van, D. Parunak, and M. Fleischer. The role of roles in designing effective agent organizations. In *Software Engineering for Large-Scale Multi-Agent Systems, LNCS 2603*, pages 27–38. Springer, 2003.

[11] A. Omicini, A. Ricci, and M. Viroli. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proc. of AAMAS 2004*, pages 286–293. ACM, 2004.

[12] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[13] J. R. Searle. *The Construction of Social Reality.* Free Press, 1995.

[14] F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *IJSEKE*, 11(3):303–328, 2001.