Enacting and Deacting Roles in Agent Programming

Mehdi Dastani, M. Birna van Riemsdijk, Joris Hulstijn,
Frank Dignum, John-Jules Ch. Meyer

Institute of Information and Computing Sciences
Utrecht University, P.O.Box 80.089, 3508 TB Utrecht,The Netherlands
tel: +31 - 30 - 253 3599
{mehdi , birna , jorish, dignum , jj }@cs.uu.nl

Abstract. In the paper we study the dynamics of roles played by agents
in multiagent systems. We capture role dynamics in terms of four op-
erations performed by agents: 'enactment', 'deactment', 'activate', and
'deactivate'. The use of these operations is motivated, in particular for
open systems. A formal semantics for these operations is provided. This
formalization is aimed at serving as a basis for implementation of role
dynamics in an agent programming language such as 3APL.

1 Introduction

Several methodologies for the development of multiagent systems have been pro-
posed to date [1, 8, 10, 14, 18]. Increasingly, these methodologies are based on or-
ganizational structures and normative concepts as cornerstones of the multiagent
systems. In these methodologies, the specification and the design of the organi-
zational structure involves two key concepts: *agent roles* and *agent types*. The
basic idea is as follows. The analysis of an application results in the specification
of an organizational structure, defined in terms of roles and their interactions.
Subsequently, at the design phase, sets of roles are translated into agent types
which constitute the system architecture. Finally, the designed system will be
implemented. We recognize that there is no consensus on the exact definition
of agent roles and agent types. In the next section we will discuss some of the
causes for the apparent difficulty to give a precise definition of roles that would
cover all its uses.

An important issue in developing multiagent systems and in particular open
multiagent systems, in which agents may enter and leave, is the need to account
for the dynamics of roles at all phases of the development methodology. The
role in which agents enter the system may determine the course of actions they
can undertake within the system and which other roles they may or may not
switch to. E.g. an agent playing a buyer role at an auction has different rights
than the seller or the auctioneer at the same auction. The dynamics of roles has
been recently studied [4, 13]. In [13], role dynamics is studied informally at the
specification level. The most important operations are *classify* and *declassify*,

which means that an agent starts and finishes to occupy a role, and *activate* and *suspend*, which means that an agent starts executing actions belonging to the role and suspends the execution of the actions. Our approach is based on similar intuitions, and therefore uses very similar operations: *enact* and *deact*[1], and *activate* and *deactivate*. In our view, enacting a role means internalizing the specification of the role, while activating a role means reasoning with the (internalized) specification of the role.

Our approach to role dynamics differs from (or complements) the approach proposed in [13] as we consider role dynamics also at the implementation level. For the implementation level, we have to explain how roles are internalized, which means that we need to assume a certain agent architecture. For this purpose, we consider cognitive agents whose behaviors are determined by reasoning (deliberating) with their mental attitudes. As we aim to describe role dynamics at the implementation level, we have to define this dynamics formally. We do this by providing the formal semantics of the operations concerning role dynamics. Based on the formal semantics for these operations, we propose programming constructs with which these can be implemented. Based on these observations we want to address the following issues.

1. Which concepts play a crucial role in each of the development stages (analysis, design and implementation) of multiagent methodologies for defining roles?
2. How can we in general specify concepts such as agent role, agent type, and role dynamics?
3. In particular, how can we extend a dedicated agent-oriented programming language with programming constructs to implement role dynamics?

To address these issues, we discuss in section 2 our views on the development of multiagent systems, and on the use of agent roles, agent types and role dynamics in specification and design. In section 3 we present a small example of an auction house to illustrate the concepts. In section 4 we present an abstract view on agent roles, agent types, and role dynamics, and relate it to an implementation in the dedicated agent-oriented programming language 3APL [6, 12].

2   Roles and Agent Types in Multiagent Methodologies

Complex system applications are analyzed by multiagent development methodologies in terms of groups, roles, agents, and their relations [10, 1, 8, 14, 18]. Although everyone has an intuitive idea about what constitutes a role, the way roles are defined and used within multiagent systems differs widely. For example, roles may be used to analyze access demands for information systems, as is done in role-based access control models (RBAC) [15], or they may be used to model aspects of stake holders in a virtual museum [2]. Different usage of a concept,

---

[1] Although 'deact' is not an English word, we think it will convey the meaning we have in mind.

means that different demands are made. However, in all approaches it seems that roles are used to identify some task, behavior, responsibility or function that should be performed within the multiagent system. Typically, roles have a descriptive and prescriptive aspect. A role describes the expected behavior and properties of an agent. For example, an agent in the buyer role is expected to want to buy something. Based on such expectations, other agents can reason about ways to interact with agents in the role. A role also prescribes the procedures and rules in an organization. In an auction, for example, one should first register as a buyer, before being allowed to bid.

We consider an agent role as a set of normative behavior rules, a set of expected objectives and a specification of the information that is expected to be available to agents playing that role. Moreover, we consider an agent type as a set of agent roles with certain constraints and assume that an agent of a certain type decides itself to enact or deact a role. We also assume that agents can have multiple enacted roles simultaneously and that an agent can enact the same role multiple times. In our approach only *one* role can be *active* at each moment in time; all other enacted roles are deactive. This is because in our view a (cognitive) agent has one single reasoning process, also called the agent's deliberation, that determines the behavior of the agent based on the enacted (internalized) roles. One single reasoning process cannot be based on two or more enacted roles at the same time. Which role should be reasoned with at each moment in time is thought to be the agent's decision.

In this paper we focus on the use of roles as a guideline for the specification, design and implementation of multiagent systems. With respect to the specification and design, we have a similar view as, for example, the Gaia methodology [18]. The details on our view on multiagent methodology can be found in [5]. The main focus of our proposed methodology is based on the distinction between closed and open multiagent systems. Our methodology aims at developing open systems in which role dynamics is an important issue. The consideration of open multiagent systems thus forms the main motivation of this work.

2.1 Open and Closed Systems

In a *closed system*, agents can be implemented to fulfill a fixed set of roles. In this setup it makes sense to design agent types as a set of roles. Not much additional structure is needed. If for example objectives from two roles could conflict, this would be a reason to alter the design and change the agent types in such a way that conflicts are avoided. So, the tasks each agent will perform are completely determined by the roles it plays. Roles themselves have no existence outside the agents in the implemented system anymore. By contrast, in an *open system* [7] agents can enter and leave such that roles have existence outside the agents in the implemented multiagent systems. In this setting, agents are not completely defined by the roles they play. Part of their behavior is determined by their own wishes and objectives, which are set and motivated from outside the multiagent system. This has a number of consequences. Roles specify the permitted and expected behavior of an agent for as long as it will be part of the system.

First, roles can be described differently in the two situations. In a closed system, roles can be described in terms of fixed tasks, or fixed motivational attitudes such as responsibilities. Although a system specification in terms of norms and roles can still be useful as a development guideline, norms and roles are not necessary at the implementation level. In an open system, the norms and roles become unavoidable at the implementation level. For example, in situations where agents cannot be trusted, the role description must provide a kind of API for the agent, to function within the multiagent system. In ISLANDER [9] this idea is made concrete by implementing roles exactly as API's through which visiting agents have to interact with other agents in the system. In more liberal systems, in which agent behavior is allowed to deviate from the expected, one could define a role in terms of norms or potential goals, together with sanctions. In this case it remains a decision of the agent how far it will comply to the norm.

Second, for closed systems, role dynamics may still be a useful development guideline to specify multiagent systems. Roles may for example be associated with certain phases in a procedure. Role dynamics can then be used to specify the progress through the procedure. But again, such notions are not necessary for the implementation of such multiagent systems. For open systems, having a proper implementation model of what it means to enact or deact a role, becomes unavoidable. Not only the order in which roles are played, but also possible conflicts and constraints need to be maintained.

Finally, in the social sciences, whether or not an agent is currently enacting a role is regarded as a *social fact* [16]. While the decision to enact/deact a role is the initiative of the agent itself, the success of performing the enactment/deactment operation is determined by the whole community. E.g. an agent entering an auction will be enlisted as a customer: the first action an agent has to perform is the enactment of the customer role. Even more important is that agents cannot decide to deact a role at any moment. For example, an agent cannot deact the customer role and leave the auction, without paying for the items it buys. So the success of a deactment action depends at least partly on external factors. Enacting and deacting are joint actions, performed by system and agent together. Although we believe that these social aspects are important, for simplicity however, we do not consider it in this paper and assume that a role change operation is always allowed. Instead, we will focus on the internal aspects of an agent enacting or deacting a role.

3   Example: Multiagent Specification and Design

In this section we present an example to illustrate the dynamics of roles and agent types in multi-agent systems. The way the example is handled is based on ideas from Islander [9] and work on skeleton programming [17]. Consider a software agent A who participates in an English auction.

1. Suppose A wants to buy a contemporary dinner table at the auction. To acquire the money, she first needs to sell her antique dinner table.

2. *A* enters the registration phase (scene) of the auction house in the role of a customer. *A*'s name, address and bank account number are registered.

3. *A* can now enter the auction phase (scene) and take up the role of the seller. The antique dinner table is then registered and a reserve price is set.

4. *A* can also enter the auction phase (scene) and take up the buyer role. When the auction lot on the contemporary dinner table starts, *A* carries out her strategy of increasing her bid until she either acquires the contemporary dinner table or reaches her personal maximum price.

5. After the auction phase, *A* can set down its seller and buyers roles, enters the payment phase (scene), and take up its customer role to settle her business. She gets a receipt for the money made by the antique table and pays for the contemporary dinner table if it has succeeded to buy it.

To analyze cases such as these, it makes sense to distinguish various *scenes*. A scene defines a social context that delimits the applicability of roles. As indicated in Figure 1, the auctioning institution of our example can be analyzed as consisting of three scenes: the registration scene, the auction scene, and the payment scene. In the registration scene, agents can enact the customer role in order to register their names, address, bank account, and other relevant information. In the auction scene, an agent can enact the seller role to register its item to be sold and set a reservation price, the buyer role to bid and buy its desired item, or the auctioneer role which controls the lots and bids. Note that the agents in the auction scene can still enact their customer role, but only in the deactivated mode. In the auction scenario, only one role (buyer, seller, or auctioneer) can be activated at the same time. Finally, agents can put down their buyer and seller roles and enter the payment scene by activating their customer role again to settle their business.

Scenes are interconnected by transitions that indicate under what conditions an agent is allowed to migrate to another scene [9, 17]. For example, in the auction scene, agents should enact the buyer, the seller, or the auctioneer role in order to enter the auction scenario. These transitions are meant to specify which activities can take place in which order. An agent can enact different roles simultaneously and this implies that an agent can be active in different scenes simultaneously. For example, in the auctioning institution, an agent can enact the customer role to enter the registration scenario (get the identity $customer_1$). After registration, it can enact the buyer role and enter the auction scenario (get the identity $buyer_1$). At this moment, the agent can activate the customer role and enter the registration scenario once again.

A role can be specified in terms of the information that becomes available to agents when they enact the role, the objectives or responsibilities that the enacting agent should achieve or satisfy, and normative rules which can for example be used to handle these objectives.
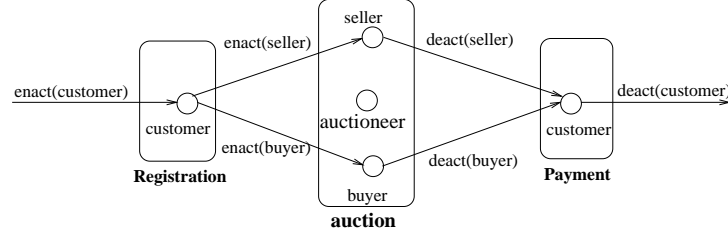
Fig. 1. Transitions between scenes

4 Formalizing Role Enactment and Role Activation

4.1 Preliminaries

In section 2, we have explained the notions of agent roles and agent types in multiagent specification and design. In this section, we formalize these concepts and describe the notions of enacting, deacting, activating, and deactivating of roles by an agent. In the following, we assume a first order language L and a set of basic actions A based on which we define the belief language L_B, the goal language L_G, and the plan language L_P.

- $L_B = \beta ::= \mathbf{B}\phi \mid \neg\beta \mid \beta \wedge \beta'$ for $\phi \in L$.
- $L_G = \kappa ::= \mathbf{G}\phi \mid \neg\kappa \mid \kappa \wedge \kappa'$ for $\phi \in L$.
- $L_P = \pi ::= \alpha \mid \beta? \mid \pi;\pi' \mid \pi + \pi' \mid \pi\|\pi' \mid \pi^*$ for $\alpha \in A,\ \beta \in L_B$

Intuitively, $\mathbf{B}\phi$ should be read as "believes ϕ", $\mathbf{G}\phi$ as "has objective ϕ", β? as "test if β", $\pi;\pi'$ as "first do π then do π'", $\pi + \pi'$ as "choose either π or π'", $\pi\|\pi'$ as "do π and π' simultaneously", and π^* as "repeat doing π". The formal semantics of these languages are not presented in this paper since it is not relevant for the purpose of this paper.

Moreover, we assume various types of rules which can be used for various purposes. For example, as we will see in the context of role specifications, these rules can be used to specify different types of norms and obligations, and in the context of agent specifications, they can be used to specify the dynamics of mental attitudes of agents such as modification or planning of objectives. For the purpose of this paper, we assume three different types of rules as specified below. The interpretation of these rules will be given when we define agent role and agent specification. Moreover, we do not claim that these types of rules are exhaustive, but believe that they make sense for the purpose of enacting and deacting of roles by agents. The three types of rules are represented by the following three sets PS (called plan selection rules), GR (called goal revision rules), and PR (called plan revision rules):

- $PS = \{\kappa \wedge \beta \Rightarrow \pi \mid \kappa \in L_G, \beta \in L_B, \pi \in L_P\}$
- $GR = \{\kappa \wedge \beta \Rightarrow \kappa' \mid \kappa, \kappa' \in L_G, \beta \in L_B\}$
- $PR = \{\pi \wedge \beta \Rightarrow \pi' \mid \pi, \pi' \in L_P, \beta \in L_B\}$

In the following, we assume that roles are abstract entities which can be instantiated whenever they are enacted. Therefore, we use $Rname$ to denote the set of names for role instantiations including a special name e for the passive role. We also use $Rules$ to indicate the set of all triples of subsets of PS, GR, and PR, i.e. $Rules = 2^{PS} \times 2^{GR} \times 2^{PR}$.

4.2 Agent Roles and Agent Types

In this approach, we assume that a role determines the information that the enacting agent should have, the objectives that it should achieve, and the norms and obligations it has to fulfill [4]. For the buyer role, the information that the enacting agent should have, includes, for example the code of the item at the auction and the starting price if it has the information of the item, i.e.
$\mathbf{B}(\ item(name, attr) \rightarrow code(name, CodeOf(attr)) \wedge price(name, PriceOf(attr))\ )$
where $CodeOf$ and $PriceOf$ are assumed to be functions that map item attributes to the code and the starting price of the item, respectively.

In this paper, we consider agent's objectives as the states that the agent *wants* to achieve. For example, the buyer role may have the goal to buy an item which can be represented as $\mathbf{G}(wantedItem(name))$. Agent norms and obligations can be considered as *states* that should be achieved (e.g. an item should be paid if it is bought), but they can also be considered as *actions* that should be performed (e.g. a buyer should register). Moreover, we consider that the norms and obligations are context-dependent and therefore conditional in nature [11]. Norms and obligations are thus represented as being conditionalized on the states. For example, the norm to ask for the information about the item that the enacting agent wants to buy can be represented by a PS rule such as $\mathbf{G}(wantedItem(name)) \Rightarrow Ask(itemInf, name)$. Note that an answer to the Ask action can cause a belief update such that $\mathbf{B}(item(name, attr))$ becomes derivable from the belief base. Note also that from this update and the information above, the enacting agent may derive the code and the starting price of the agent. Moreover, an obligation to pay for a bought item can be represented by a GR rule as follows: $\mathbf{B}(bought(item)) \Rightarrow \mathbf{G}(pay(item))$.

Definition 1. *(Role) A role is a tuple $\langle \sigma_i, \gamma_i, \omega_i \rangle$, typically denoted by r, where $\sigma_i \in L_B$ specifies the information that an agent receives when enacting this role, $\gamma_i \in L_G$ specifies the objectives to be achieved by the agent that enacts this role, and $\omega_i \in Rules$ be a triple consisting of rules representing conditional norms and obligations.*

We assume that the objectives γ_i in the above definitions are achievement goals. Maintenance goals can be defined in terms of normative rules of the form $\neg\kappa \wedge \top \Rightarrow \kappa$ which means that goals κ should be adopted whenever κ is not the case. A role can be incoherent in the sense that it may be specified in terms of inconsistent beliefs and goals. Also, normative rules that are ascribed to a role may suggest the adoption of inconsistent objectives. One may therefore introduce coherence conditions to exclude these cases.

Definition 2. *(Role coherency) Let* $\omega_i = (\omega^{PS}, \omega^{GR}, \omega^{PR}) \in Rules$. *A role* $r = \langle \sigma_i, \gamma_i, \omega_i \rangle$ *is coherent, denoted as coherent(r), iff:*

1. $\sigma_i \not\models \bot$ *: consistent beliefs*
2. $\gamma_i \not\models \bot$ *: consistent objectives*
3. $\sigma_i \not\models \gamma_i$ *if* $\top \not\models \gamma_i$ *: non-trivial objectives are not achieved*
4. $(\bigwedge_{(\kappa \wedge \beta \Rightarrow \kappa') \in \omega^{GR}} \kappa') \not\models \bot$ *: potential objectives are mutually consistent*
5. $\forall (\kappa \wedge \beta \Rightarrow \kappa') \in \omega^{GR} \; : \; \kappa' \wedge \gamma_i \not\models \bot$ *: potential objectives are consistent with*
                                                        role's objectives

Note that clause 4 in this definition is very strong in that it requires that all potential objectives should be mutually consistent. This requirement can be dropped resulting in a less restricted notion of coherence.

Roles can be mutually inconsistent since they may have contradictory information and objectives. Below, we define the notion of role consistency.

Definition 3. *(Role consistency) Two roles* $r = \langle \sigma_1, \gamma_1, \omega_1 \rangle$ *and* $r' = \langle \sigma_2, \gamma_2, \omega_2 \rangle$ *are consistent, denoted as consistent(r, r'), iff their 'combined role' is coherent, i.e.*

$$consistent(r, r') \Leftrightarrow coherent(\langle \sigma_1 \wedge \sigma_2, \gamma_1 \wedge \gamma_2, \omega_1 \oplus \omega_2 \rangle)$$

where $(R_1, \ldots, R_n) \oplus (R_1', \ldots, R_n') = (R_1 \cup R_1', \ldots, R_n \cup R_n')$.

Proposition 1. *An agent role r is coherent iff it is consistent with itself, i.e.*

$$coherent(r) \Leftrightarrow consistent(r, r)$$

An agent can enact different roles during its execution (one actively at a time) and enacting a role influences its mental attitudes. As explained in section 2, the type of the agent determines the roles that the agent can enact. Therefore, we require that the roles that an agent can enact should be mutually consistent since these roles influence the agent's mental attitudes.

Definition 4. *(Agent Type) Let $\mathcal{R}$ be the set of agent roles. An agent type t with respect to $\mathcal{R}$ is a consistent subset of agent roles, i.e. $t \subseteq \mathcal{R}$ such that $\forall r, r' \in t \; : \; consistent(r, r')$.*

Proposition 2. *All agent roles from an agent type $t \subseteq \mathcal{R}$ are coherent, i.e.*

$$\forall r \in t : coherent(r)$$

4.3 Role Enacting and Role Deacting Agents

In this paper, we assume that role enacting agents have their own mental attitudes consisting of beliefs, goals, plans, and rules that may specify their conditional mental attitudes as well as how to modify their mental attitudes. In addition, a role enacting agent is assumed to enact a set of roles among which only one of them is active at each moment in time; all other enacted roles are inactive. The reason for assuming one active role at each moment of time is explained in section 2. Therefore, role enacting agents have distinct objectives and

rules associated to the active role it is enacting, and sets of distinct objectives and rules adopted from enacted but inactive roles. The roles enacted by an agent are instantiations of the roles specified in t. This can be compared to objects which are instantiations of classes. It is therefore possible that one role from t is enacted and instantiated several times. We call an agent with its own mental attitudes, an active role instantiation, a set of inactive role instantiations, and a type, a *role enacting agent*.

Definition 5. *(role enacting agent: rea) Let $\gamma_a \in L_G$, $\gamma_r \in L_G \times Rname$, and $\gamma \subseteq L_G \times Rname$. Let $\Pi_a \subseteq L_P \times L_G$ and $\Pi_r \in 2^{L_P \times L_G} \times Rname$, $\Pi_s \subseteq 2^{L_P \times L_G} \times Rname$. Let $\omega_a \in Rules$, $\omega_r \in Rules \times Rname$, $\omega \subseteq Rules \times Rname$, and $e \in Rname$ be a special role instantiation name for passive role. Then, a role enacting agent is a tuple $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle$, where:*

- *$\sigma \in L_B$ specifies rea's beliefs*
- *$\Gamma = (\gamma_a, \gamma_r, \gamma)$ specifies rea's objectives*
- *$\Pi = (\Pi_a, \Pi_r, \Pi_s)$ specifies rea's plans*
- *$\Omega = (\omega_a, \omega_r, \omega)$ specifies rea's rules*
- *$t \subseteq \mathcal{R}$ s.t. $\forall r \in t : consistent(\langle \sigma, \gamma_a, \omega_a \rangle, r)$ specifies rea's type.*

A passive-role enacting agent (p-rea) is defined as a rea where $\Gamma = (\gamma_a, (\top, e), \gamma)$, $\Pi = (\Pi_a, (\emptyset, e), \Pi_s)$, and $\Omega = (\omega_a, ((\emptyset, \emptyset, \emptyset), e), \omega)$.

In the above definition, γ_a and ω_a specify the agent's own objective and rules, respectively. Moreover, γ_r and ω_r specify respectively the objective and rules associated to the active role that the agent enacts, and γ and ω are sets of objectives and sets of rules of the enacted roles which are not active, respectively. Finally, Π_a specifies agent's own plans, Π_r specifies the plans that are generated by the active role, and Π_s specifies the plans of enacted but inactive roles. Note that an objective is associated with each plan to indicate the (initial) purpose of that plan. Also, a role instantiation name is associated with the objectives in γ_r and γ, to the plans in Π_r and Π_s, and with the sets of rules in ω_r and ω. Finally, note that the last clause ensures that agent roles are consistent with the mental attitudes of the agent. As for roles, one can also define coherency for rea's.

Definition 6. *(coherent rea) Let $r_0, r_1, \ldots, r_n \in Rname$, $\gamma_0, \gamma_1, \ldots, \gamma_n \in L_G$, and $\omega_0, \omega_1, \ldots, \omega_n \in Rules$ for $n \geq 0$. The rea $\langle \sigma, (\gamma_a, \gamma_r, \gamma), \Pi, (\omega_a, \omega_r, \omega), t \rangle$ is coherent iff its belief is consistent and it consists of corresponding objective/rules pairs from the enacted (active and inactive) roles each with a unique role instantiation name, i.e. iff the following conditions hold:*

1. *$\sigma \not\models \bot$*
2. *$\gamma_r = (\gamma_0, r_0)$ & $\omega_r = (\omega_0, r_0)$*
3. *$\gamma = \{(\gamma_1, r_1), \ldots, (\gamma_n, r_n)\}$ & $\omega = \{(\omega_1, r_1), \ldots, (\omega_n, r_n)\}$ & $r_i \neq r_j$ for $1 \leq i \neq j \leq n$*

Note that we use r_i, r_j as typical denotations for role instantiation names, and r, r' as typical denotations to role specifications. The first clause states that

the belief base of a rea should be consistent, the second states that objectives and rules of the active role should be from one and the same role instantiation, the third states that there should be a bijection between objectives and rules of inactive roles, and the last clause states that the role instantiation names used in a rea should be unique. Note that the notion of coherence can be made stronger by demanding that the agent's own objective does not conflict with the objectives of the enacted (active and inactive) roles, i.e. by adding the following condition: $\gamma_a \wedge \gamma_i \not\models \bot$ for $0 \leq i \leq n$. Note that a passive-role enacting agent (p-rea) is a coherent rea.

In our view, enacting a role by an agent means that the agent adopts the role (i.e. it adopts the information, objectives, and rules that are associated with the role) and uses a name to refer to the instantiation of this role. Enacting a role can be specified by a function that maps rea's, roles, and role instantiation names to rea's. This function is defined on rea's in general, rather than on coherent rea's. In proposition 3 below, we relate this function and the notion of coherent rea's.

Definition 7. *(Role enacting function) Let $\mathcal{S}$ be the set of rea's, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in \mathcal{S}$, $\mathcal{R}$ be the set of roles, $\langle \sigma_i, \gamma_i, \omega_i \rangle \in \mathcal{R}$, and $r_i \in Rname$ be a role instantiation name. The role enacting function $\mathcal{F}_{enact} : \mathcal{S} \times \mathcal{R} \times Rname \rightarrow \mathcal{S}$ is defined as follows:*

$$\mathcal{F}_{enact}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \ , \ \langle \sigma_i, \gamma_i, \omega_i \rangle \ , \ r_i) = \langle \sigma \wedge \sigma_i, \Gamma', \Pi, \Omega', t \rangle$$

where $\Gamma = (\gamma_a, \gamma_r, \gamma)$ and $\Gamma' = (\gamma_a, \gamma_r, \gamma \cup \{(\gamma_i, r_i)\})$,
$\Omega = (\omega_a, \omega_r, \omega)$ and $\Omega' = (\omega_a, \omega_r, \omega \cup \{(\omega_i, r_i)\})$.

An agent may decide to deact a role which means that the agent stops enacting the role. In our view, the agent that deacts a role will remove the objective and plans adopted by enacting the role. Note that the plans can be generated during the enactment of the role.

Definition 8. *(Role deacting function) Let $\mathcal{S}$ be the set of rea's, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in \mathcal{S}$, and $r_i \in Rname$ be a role instantiation name. The role deacting function $\mathcal{F}_{deact} : \mathcal{S} \times Rname \rightarrow \mathcal{S}$ is defined as follows:*

$$\mathcal{F}_{deact}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \ , \ r_i) = \langle \sigma, \Gamma', \Pi', \Omega', t \rangle$$

where $\Gamma = (\gamma_a, \gamma_r, \gamma \cup \{(\gamma_i, r_i)\})$ and $\Gamma' = (\gamma_a, \gamma_r, \gamma)$,
$\Pi = (\Pi_a, \Pi_r, \Pi_s \cup \{(X, r_i)\})$ and $\Pi' = (\Pi_a, \Pi_r, \Pi_s)$,
$\Omega = (\omega_a, \omega_r, \omega \cup \{(\omega_i, r_i)\})$ and $\Omega' = (\omega_a, \omega_r, \omega)$.

In the following, we say that a role instantiation name r_i does (or does not) occur in a rea $s = \langle \sigma, (\gamma_a, \gamma_r, \gamma), \Pi, (\omega_a, \omega_r, \omega), t \rangle$ if r_i does (or does not) occur in the pair γ_r and ω_r and does (or does not) occur in the pairs contained in γ and ω.

Proposition 3. *Let $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle$ be a coherent rea, $r \in t$, and $r_i \in Rname$. Then, the rea $\mathcal{F}_{enact}(s, r, r_i)$ is coherent if r_i does not occur in s, and the rea $\mathcal{F}_{deact}(s, r_i)$ is coherent.*

Note that the deacting function can only deact an inactive role. Note also that for some $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle$, $r \in t$ and $r_i \in Rname$ the following hold:

$$\mathcal{F}_{deact}(\mathcal{F}_{enact}(s, r, r_i), r_i) \neq s \quad \text{and} \quad \mathcal{F}_{enact}(\mathcal{F}_{deact}(s, r_i), r, r_i) \neq s$$

For example, consider $s = \langle p, (\gamma_a, \gamma_r, \gamma), \Pi, (\omega_a, \omega_r, \omega), t \rangle$, $\langle q, \gamma_i, \omega_i \rangle$ be a role, and $r_i \in Rname$ which does not occur in s. Then,
$\mathcal{F}_{enact}(s, \langle q, \gamma_i, \omega_i \rangle, r_i) = \langle p \wedge q, (\gamma_a, \gamma_r, \gamma \cup \{(\gamma_i, r_i)\}), \Pi, (\omega_a, \omega_r, \omega \cup \{(\omega_i, r_i)\}), t \rangle$
and $\mathcal{F}_{deact}(\mathcal{F}_{enact}(s, \langle q, \gamma_i, \omega_i \rangle, r_i), r_i) = \langle p \wedge q, (\gamma_a, \gamma_r, \gamma), \Pi, (\omega_a, \omega_r, \omega), t \rangle \neq s$.

However, starting with a role enacting agent whose belief base entails the belief base of a role, then enacting followed by deacting of the role by the same agent gives the identity function.

Proposition 4. *Let rea s' be of the form $\mathcal{F}_{deact}(\mathcal{F}_{enact}(s, r, r_i), r_i)$ and rea s'' be of the form $\mathcal{F}_{enact}(\mathcal{F}_{deact}(s, r_i), r, r_i)$, for the role $r \in t$ and $r_i \in Rname$. Then,*

$$\mathcal{F}_{deact}(\mathcal{F}_{enact}(s', r, r_i), r_i) = s' \quad and \quad \mathcal{F}_{enact}(\mathcal{F}_{deact}(s'', r_i), r, r_i) = s''$$

4.4 Activating and Deactivating Roles

In our view, enacting a role does not imply activating the role. However, since enacting a role updates the belief base of the rea, the enacted role will indirectly influence the behavior of the role enacting agent. In order to direct the role enacting agent to achieve the role's objectives, the enacted role should be activated. In fact, activating a role is selecting and processing it. For this reason, we introduce two new functions for *activating* and *deactivating* agent roles. The role activating function maps passive-role enacting agents to role enacting agents. The objectives, plans, and rules of the enacted role become active entities and will affect the behavior of the role enacting agent.

Definition 9. *(Role activating function) Let $\mathcal{S}$ be the set of rea's, $\mathcal{S}^e$ be the set of passive-role enacting agents, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in \mathcal{S}^e$, $\mathcal{R}$ be the set of roles, $\langle \sigma_j, \gamma_j, \omega_j \rangle \in \mathcal{R}$, and $r_i \in Rname$. The role activating function $\mathcal{F}_{activate} : \mathcal{S}^e \times \mathcal{R} \times Rname \to \mathcal{S}$ is defined as follows:*

$$\mathcal{F}_{activate}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \ , \ \langle \sigma_j, \gamma_j, \omega_j \rangle \ , \ r_i) = \langle \sigma \wedge \sigma_j, \Gamma', \Pi', \Omega', t \rangle$$

where
$\Gamma = (\gamma_a, (\top, e), \gamma \cup \{(\gamma_i, r_i)\})$ *and* $\Gamma' = (\gamma_a, (\gamma_i, r_i), \gamma)$,
$(\Pi = (\Pi_a, (\emptyset, e), \Pi_s)$ *and* $\forall X \subseteq L_P \times L_G\ (X, r_i) \notin \Pi_s$ *and* $\Pi' = \Pi)$ *or*
$(\Pi = (\Pi_a, (\emptyset, e), \Pi_s \cup \{(X, r_i)\})$ *and* $\Pi' = (\Pi_a, (X, r_i), \Pi_s))$,
$\Omega = (\omega_a, ((\emptyset, \emptyset, \emptyset), e), \omega \cup \{(X, r_i)\})$ *and* $\Omega' = (\omega_a, (X, r_i), \omega)$.

Note that the second argument of the role activating function is a role specification while we only use the information component of the role specification, i.e. σ_j. Alternatively, we can specify the role activating function without agent specification as argument, but then we have to modify the rea specification to

represent the information associated to the inactive roles. The second condition in the above definition is relevant when an enacted role is activated for the first time. Note that the roles do not contain plans in their specifications and enacting them do not add any plan to a rea. The plans related to a role can be generated only by activating the role. The third condition guarantees that the plans of an already activated role, which have been stored by their deactivation (see definition 10 for deactivation of roles), are activated again.

The role deactivating function, to the contrary, t maps role enacting agents to passive-role enacting agents. In fact, the activated enacting role may consist of objectives that are not achieved and plans that are not executed. These entities are saved and can be activated once again.

Definition 10. *(Role deactivating function) Let $\mathcal{S}$ be the set of rea's, $\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in \mathcal{S}$, $\mathcal{S}^e$ be the set of passive-role enacting agents, and $r_i \in Rname$. The role deactivating function $\mathcal{F}_{deactivate} : \mathcal{S} \times Rname \rightarrow \mathcal{S}^e$ is defined as follows:*

$$\mathcal{F}_{deactivate}(\langle \sigma, \Gamma, \Pi, \Omega, t \rangle \ , \ r_i) = \langle \sigma, \Gamma', \Pi', \Omega', t \rangle$$

where
$\Gamma = (\gamma_a, (\gamma_i, r_i), \gamma)$ *and* $\Gamma' = (\gamma_a, (\top, e), \gamma \cup \{(\gamma_i, r_i)\})$,
$\Pi = (\Pi_a, (X, r_i), \Pi_s)$ *and* $\Pi' = (\Pi_a, (\emptyset, e), \Pi_s \cup \{(X, r_i)\})$,
$\Omega = (\omega_a, (X, r_i), \omega\})$ *and* $\Omega' = (\omega_a, ((\emptyset, \emptyset, \emptyset), e), \omega \cup \{(X, r_i)\})$.

Proposition 5. *Let $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle$ be a passive-role enacting rea (p-rea), $r \in t$, and $r_i \in Rname$ occurs in s. Then, the rea's $\mathcal{F}_{activate}(s, r, r_i)$ and $\mathcal{F}_{deactivate}(s, r_i)$ are coherent.*

A role enacting agent can be activated and deactivated. Note that there exists a rea $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle$ and $s^e = \langle \sigma', \Gamma', \Pi', \Omega', t' \rangle$ in which $r_i, r_i' \in Rname$ occurs, respectively, such that the following hold:

$$\mathcal{F}_{activate}(\mathcal{F}_{deactivate}(s, r_i), r, r_i) \neq s \quad \text{for } r \in t$$

$$\mathcal{F}_{deactivate}(\mathcal{F}_{activate}(s^e, r', r_i'), r_i') \neq s^e \quad \text{for } r' \in t'$$

In general, the behavior of the recursive applications of activating and deactivating functions is characterized by the following proposition.

Proposition 6. *Let the rea s be of the form $\mathcal{F}_{activate}(s^e, r, r_i)$ where $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle$, $r \in t$, and $r_i \in Rname$, then*

$$\mathcal{F}_{activate}(\mathcal{F}_{deactivate}(s, r_i), r, r_i) = s$$

The enacting agent can enact the role in various ways. For example, the agent may prefer to achieve the objectives adopted from the role before aiming to achieve its own objective, or otherwise it may prefer to achieve its own objective first. The exact way to enact a role should be determined either beforehand or during the execution of the agent.

5 Implementation of Roles

Like other programming languages, an agent programming language should provide data structures to specify the (initial) state, and a set of programming constructs to specify how the states should evolve. In the case of programming languages for cognitive agents the data structures consist of mental attitudes such as beliefs, goals, and plans, and the specification of their dynamics is captured by the modification rules. The programming constructs consist of a set of basic operations, which are defined on mental attitude and the rules, and a set of operators to compose complex programming constructs in terms of basic operations. The program that specifies the operations on these entities is usually called the deliberation program, deliberation cycle, or decision making mechanism of agents [3].

In general, there are two ways to implement the enactment and deactment of roles by cognitive agents. The first approach is to introduce two special actions that can be invoked in the agent's plan and which, when executed, realize the enactment and deactment of roles. In this approach, the agent will enact and deact a role according to its plans that are conditionalized for example by its beliefs or goals. For example, an agent *buyer* may have the plan if $\mathbf{B}(\texttt{registered(me)})$ then $\texttt{enact}(r_{\texttt{buyer}}, \texttt{buyer}_1)$ which, when executed, updates the *buyer* according to the instantiation of the role $r_{\texttt{buyer}}$ (denoted by $\texttt{buyer}_1$) if he believes that he is already registered. Also, one may specify the goals and rules, which specify a role in such a way that the agent will execute the deact action when the objectives of the agent are achieved. For example, in our auction example, the role $r_{\texttt{buyer}}$ (instantiated and denoted by the role name $\texttt{buyer}_1$) may contain a rule $\mathbf{B}\ \texttt{bought(item)} \Rightarrow \texttt{deact}(\texttt{buyer}_1)$. This rule indicates that whenever the role enacting agent believes that it bought the item, then he should deact the buyer role $\texttt{buyer}_1$.

The second approach is to introduce two basic programming (deliberation) operations which, when executed, result in enacting or deacting of agent roles. These and other operations such as selecting goals and plans, executing plans, or applying modifications rules constitute the agent's deliberation program. For example, a deliberation program can consist of selecting and enacting a role (in this case based on goal $\mathbf{G}(buy(item))$) before starting an iteration in which a goal of the agent is selected and planned and the plan executed. In this iteration, the rules may also be selected and applied to modify the goals and plans of the agent. Let $\texttt{enact}(r_{\texttt{buyer}}, \texttt{buyer}_1)$ and $\texttt{deact}(\texttt{buyer}_1)$ be deliberation operations for enacting and deacting the instantiation of the role buyer, respectively. Then, the following illustrates a deliberation program in which the agent first selects which role to enact, then enact the role, and finally deact it.

```
1- If G(buy(item)) then
2- BEGIN
3-    enact(r_buyer, buyer_1);activate(r_buyer, buyer_1);
4-    While goalbase ≠ ⊤ do Select_goal(G);Plan(G,P);Execute(P) od;
5-    deactivate(buyer_1);deact(buyer_1)
6- END
```

In both approaches, the enactment and deactment of roles result in a modification of the role enacting agent as specified in definitions 7 and 8. According to these definitions, enacting a role results in adoptions of beliefs, goals, and rules, and deacting it results in removal of goals, plans, and rules. The choice for one of these two approaches will be based on a pragmatic consideration and is a methodological issue [5]. For example, one should consider if role modification is a part of the agent's mental attitudes or is it an issue of an agent decision making process.

5.1 Semantics of Enact and Deact Operations

The semantics of programming languages can be specified in terms of updates (or transitions) of states (agent specification) based on programming operations. For example, we have provided in [6] the operational semantics of 3APL, which is a programming language for cognitive agents. In this section, we assume an arbitrary cognitive agent programming language for which operational semantics is defined. We sketch how the semantics of this language can be modified as the result of extending the language with enact and deact operations. In particular, we explain which parts of the existing semantics should be modified, and how the semantics of the enact and deact operations should be defined.

In section 4, we have defined an agent specification in such a way to allow agents to have an explicit representation of the role they enact. In particular, we have defined the agent's goal base and rule bases as tuples to have a distinguished representations of the objectives and rules of the agent itself and the objectives and rules that specify the active and inactive roles. The fact that the goal base and the rule base are tuples raises the question how to verify whether a goal is derivable from the goal base and how to select a rule from the rule base. Given $\langle \gamma_a, \gamma_r, \gamma \rangle$ as the goal base of a role enacting agent and κ a goal, the first question can be answered by verifying if the goal is derivable from the conjunction of the goal bases, i.e. $\gamma_a \wedge \gamma' \models \kappa$. Given $\langle \omega_a, \omega_r, \omega \rangle$ as the rule base of the role enacting agent (with active role r_i), a rule can be selected from the set $\omega_a \cup \omega'$. We assume that rules will be selected from the set of rules based on orderings defined on ω_a and ω', and based on a selection criterion. An example of selection criteria is the attitude of the role enacting agent, e.g. social (first select from ω' before selecting from ω_a), or selfish (first select from ω_a before selecting from ω').

In the following, we specify the update of agent states based on the enacting and deacting operations. The provided updates can be used to define transitions if the semantics of the programming language is an operational semantics. In the following, we use the semantic function $Sem(\alpha, s) = s'$ to indicate that the state s' is the result of updating the state s through operation α.

Definition 11. *Let $\mathcal{S}$ be the set of role enacting agents, $s = \langle \sigma, \Gamma, \Pi, \Omega, t \rangle \in \mathcal{S}$, $\mathcal{R}$ be the set of roles, $r \in \mathcal{R}$, $r_i \in Rname$, and $\omega' \in Rules$. Let $\mathcal{F}_{enact}, \mathcal{F}_{deact}, \mathcal{F}_{activate}$, and $\mathcal{F}_{deactivate}$ as defined in definitions 7, 8, 9, and 10, respectively. The semantics of the operations $OP = \{$ enact(r, r_i), deact(r_i), activate(r, r_i),*

`deactivate(`r_i`)}`, *is captured by the function* $Sem : OP \times \mathcal{S} \rightarrow \mathcal{S}$, *defined as follows:*

$Sem(\text{enact}(r, r_i), s) = \mathcal{F}_{enact}(s, r, r_i)$ *for* $r \in t$

$Sem(\text{deact}(r_i), s) = \mathcal{F}_{deact}(s, r_i)$ *for* $\Omega = (\omega_a, \omega_r, \omega)$ & $(\omega', r_i) \in \omega$

$Sem(\text{activate}(r, r_i), s) = \mathcal{F}_{activate}(s, r, r_i)$ *for* $\Omega = (\omega_a, \omega_r, \omega)$ & $(\omega', r_i) \in \omega$

$Sem(\text{deactivate}(r_i), s) = \mathcal{F}_{deactivate}(s, r_i)$ *for* $\Gamma = \langle \gamma_a, (\gamma', r_i), \gamma \rangle$

$and \ \Omega = \langle \omega_a, (\omega', r_i), \omega \rangle$

Based on this semantics for the proposed programming instructions and assuming that other programming instructions maintain the coherence of rea's, then we can formulate the following safety proposition.

Proposition 7. *(safety) Let s be a coherent rea and P be an agent program consisting of a set of programming instructions among which those related to enacting and activating roles as suggested in definition 11. Let the following conditions hold:*

- *each instruction* `deact(`r_i`)` *is preceded by an instruction* `enact(`r, r_i`)` *between which* r_i *is used uniquely*
- *each instruction* `deactivate(`r_i`)` *is preceded by only one instruction* `activate(`r, r_i`)` *between which* r_i *is used uniquely, and no* `activate(`r, r_i`)` *is preceded by another* `activate(`r', r_j`)`
- *all other programming instructions maintain coherence of rea's*

Then, if the program P is executed on rea s, the resulted rea after the execution of P is coherent.

6   Future Research and Concluding Remarks

In this paper we have argued for the importance of enactment/deactment of roles by agents in multiagent programming, in particular when dealing with open systems where the match between the agents in the system and the roles to be played is not fixed but changing dynamically. Since we furthermore believe that an agent can only be actively engaged in one role at a time, we have also proposed an activate/deactivate mechanism for roles. We have provided a formal semantics of the enactment and deactment as well as the activate and deactivate operations. Since this formalization is conceptually based on the notion of cognitive agents (and employs concepts used in the semantics of an agent language such as 3APL in particular), we claim that the implementation of the proposed mechanism by agent-oriented programming languages is straightforward.

References

1. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, to appear.

2. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the TROPOS project. *Information Systems*, 27:365–389, 2002.

3. M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation. In *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*. 2003.

4. M. Dastani, V. Dignum, and F. Dignum. Role-assignment in open agent societies. In *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*. 2003.

5. M. Dastani, J. Hulstijn, F. Dignum, and J.-J. Meyer. Issues in multiagent system development. In *Proceedings of The Third Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04)*. New York, USA, 2004.

6. M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Meyer. A programming language for cognitive agents: Goal directed 3APL. In M. Dastani, J. Dix, A. E. Fallah-Seghrouchni, and D. Kinny, editors, *Proceedings of the First Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS03)*. 2003.

7. P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agent World II*, LNAI 2203. Springer Verlag, Berln, 2001.

8. V. Dignum. *A Model for Organizational Interaction, based onAgents, founded in Logic*. PhD thesis, University of Utrecht, 2003.

9. M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *First Interantional Joint Conference on Autonoumous Agents and Multiagent Systems (AAMAS'02)*, pages 1045 – 1052. ACM Press, 2002.

10. J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, LNCS, pages 214–230. Springer Verlag, 2003.

11. B. Hansson. An analysis of some deontic logics. *Nous*, 3:373–398, 1969.

12. K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

13. J. Odell, H. V. D. Parunak, S. Brueckner, and J. Sauter. Temporal aspects of dynamic role assignment. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, LNCS, pages 201–213. Springer Verlag, 2003.

14. A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In *AOSE*, pages 185–193, 2000.

15. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2), 1996.

16. J. Searle. *The Construction of Social Reality*. The Free Press, New York, 1995.

17. W. W. Vasconcelos, J. Sabater, C. Sierra, and J. Querol. Skeleton-based agent development for electronic institutions. In *First Interantional Joint Conference on Autonoumous Agents and Multiagent Systems (AAMAS'02)*, pages 696–703. ACM Press, 2002.

18. M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.