AAECC (2005) 16: 151–178
DOI 10.1007/s00200-005-0174-z

Andrea Omicini · Alessandro Ricci ·
Mirko Viroli

An algebraic approach for modelling organisation, roles and contexts in MAS

Received: 11 March 2004 / Revised: 1 August 2004
Published online: 7 July 2005
© Springer-Verlag 2005

Abstract Governing the ever growing complexity of artificial systems on the one hand requires a number of expressive abstractions and different levels of interpretation, on the other hand suggests the adoption of formal / mathematical tools to (at least partially) model and predict the system behaviour. By adopting agent-oriented abstractions as the starting point, we argue that organisation, coordination and security all insist on the same conceptual space – that is, static / dynamic relations / interactions among agents –, which also represents one of the main sources of complexity for MAS, and for artificial systems in general, as well.

The notion of ACC (agent coordination context) is used in this paper as the unifying core abstraction of a framework that encompasses all such issues, promoting the integration of organisation, coordination, and security. Such a framework, called RBAC-MAS, is expressed through a process algebraic model which integrates the classic organisational issues of role-based models (like RBAC) and the more recent works on interaction and coordination in MAS.

1 Introduction

The many ways of interaction among system components represent the main source of complexity within non-trivial systems of any sort. This is why so many research and technology efforts in the area of computational systems have been devoted in the last years to the issue of "harnessing the space of interaction" between components [35]: coordination, organisation, security, computer networks, workflow management, computer-supported cooperative work, are only some of the many tags that have labelled the scientific work in such a broad area of interest.

A. Omicini (✉) · A. Ricci · M. Viroli
DEIS, Dipartimento di Elettronica, Informatica e Sistemistica, Alma Mater Studiorum, Università di Bologna, via Venezia 52, 47023, Cesena, Italy
E-mail: andrea.omicini@unibo.it

Coordination can be considered as one of the main topics in the area: as such, it has been the subject of several investigations within a multiplicity of different research areas, and, correspondingly, has been differently conceived and defined [31]. Quite generally, coordination can be defined as the management of interaction – interaction among the components of a system, whichever the components, whichever the system. According to this general acceptation, coordination comes hand in hand with two other key issues in complex (computational) systems: *security* and *organisation*.

Security represents in some sense a dual aspect with respect to coordination: as discussed in [6], coordination could be seen as the constructive counterpart of security in the dynamics of interaction. That is, whereas security focuses on *preventing* undesired / incorrect system behaviours – which may result in problems like denial of services or unauthorised access to resources – coordination is mostly concerned with *enabling* desirable / correct system behaviours – typically the meaningful, goal-directed interaction between different system components.

On the other hand, organisation strictly relates to both security and coordination. Apart from organisation abstractions typically used in the security field – such as *roles* in RBAC approaches [33], used as both organisation and security abstractions –, organisation deals with the static aspects of interaction, while coordination (and security, dually) mainly deals with the dynamics of interaction. Roughly speaking, the organisation of a system (whether a human or an artificial one) defines the admissible interactions among system components at design time, while coordination and security mostly deal with governing interaction at execution time. Then, it comes not as unexpected that organisational sciences have provided some powerful conceptual tools for effectively interpreting and framing the role of coordination within complex systems of any sort: this is the case, for instance, of Activity Theory, as discussed in [32].

As discussed in [23], *infrastructures* are the most natural *loci* where to embody a uniform framework accounting for organisation, coordination and security altogether. In order to support fundamental engineering practices like incremental development, on-line verification, corrective / adaptive / evolutive maintenance, in fact, infrastructures are today required to provide systems at run time with the same abstractions used at the design stage. So, for instance, the design abstractions typically used to model organisation, coordination and security – such as roles, access lists, and coordination media like monitors or tuple spaces – are nowadays to be provided as run-time abstractions by the system infrastructures.

By their very nature, infrastructures intrinsically encapsulate key portions of systems, often in charge of critical system behaviours; as a result, infrastructures also represent an effective approach to the general problem of formalisability of complex systems. Formal specification of infrastructures – and in particular of the abstractions they embody – is then a key issue for the discovery and proof of critical system properties.

Along the above lines, in this paper we focus on Multi-Agent Systems (MAS) as a relevant class of complex computational systems. In Section 2, we first introduce the notion of ACC (Agent Coordination Context) [22] as an infrastructure abstraction encompassing organisation, security and coordination, then we provide some examples of the ACC expressive power. Then, in Section 3 we first give an overview of RBAC (Role-Based Access Control) models, then we show how a role-based approach to security can be adapted and extended to agent-based organisations, by informally presenting our RBAC-MAS model. The formal discussion of both the ACC notion and the full RBAC-MAS model is delegated to Section 4, where we exploit the potential of algebraic approaches to denote MASs and their evolution; first we introduce our ontology (what is an agent, a MAS, an organisation, an agent action, an ACC, and all the essential ingredients of RBAC-MAS) in terms of the elements of a process algebra, then we present the transition systems that define the semantics of ACC and RBAC-MAS. Finally, Section 5 provides for final remarks and conclusion.

2 Agent coordination contexts

Introduced in [22], the notion of ACC (agent coordination context) aims at facing the many static and dynamic issues that arise from the relation between the individual agent and the MAS as a whole. In particular, ACCs have been conceived so as to uniformly frame the agent-vs-MAS issues that are usually expressed in terms of organisation, coordination and security.

In the remaining of this section, we first introduce the notion of ACC by moving from its motivations and conceptual foundations, then we discuss ACC as an infrastructure abstraction. Finally, we provide some examples of the ACC use and expressive power, by anticipating some of the syntax defined in the rest of the paper. The formal definition of the ACC notion is in fact delegated to Section 4, where a general framework for modelling MASs with ACCs and RBAC-MAS is introduced as a process-algebraic specification.

2.1 ACC basics

Generally speaking, an ACC is meant to represent the conceptual boundary between the agent and the environment (or, the rest of the MAS), encapsulating the interface toward the environment from the agent viewpoint. As an interface, the ACC both (i) works as a model of the environment for the agent, and (ii) enables and rules the agent interactions with the environment. The ACC abstraction models the presence of an agent within a multi-agent organisation, by defining its admissible actions (including perception) with respect to the organisation resources, as well as its admissible communication acts toward the other agents belonging to the organisation.

An ACC works both as a specification for the rules that constrain agent interaction with the environment, and also as a means to enact such rules. The ACC is meant to encapsulate such rules, to enforce them, and also to make them available for agent run-time inspection and, possibly, meta-level reasoning over the MAS state and behaviour.

In order to provide the reader with a better intuition of the concept, an ACC could be basically thought as a *control room* made available to each agent entering a MAS [22]. According to this metaphor (sketched in Fig. 1), an agent entering a new organisation is assigned its own control room. The control room offers the agent a set of admissible inputs (lights, screens, ...) and admissible outputs (buttons, cameras, ...), which are the only way the agent can perceive

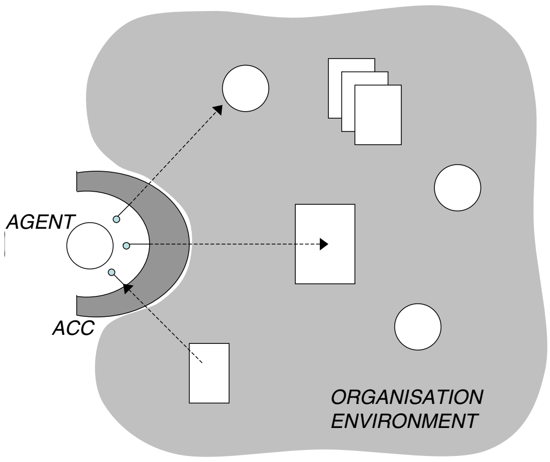

Fig. 1 ACC as a control room [22]

the environment, as well as the only way the agent can interact with the environment and affect it. How many input and output devices are available to an agents, of what sort, and for how much time – this is what defines the control-room *configuration*, that is, out of metaphor, the specific ACC configuration. On the other hand, the control room (so, the ACC) is the only way the agent is perceived by its surrounding environment, that is, by the rest of the MAS.

For instance, in the context of MASs based on direct communication models, such as KQML [17] or FIPA ACL [12], the ACC would work by enabling and ruling agent speech acts and conversations. When exploited in the context of agent coordination infrastructures such as TuCSoN – as studied in [24] – an ACC works by governing agent interactions with the coordination media (such as tuple centres), expressed in terms of primitives of the underlying coordination model (such as `in`, `out`, `rd`, ...).

In general, the ACC is meant to enable and rule agent actions of any sort. According to that, the general model for ACC cannot assume any specific action model – which would depend on the specific MAS – and relies instead on a generic, non-specific notion of agent action. Dually, any given MAS defines its own specific action model, which ACCs are then required to enable and rule: so the specific MAS action model determines the specific ACC model on a case-by-case basis.

2.2 Organisation, security, coordination & the ACC

2.2.1 Organisation

Two basic stages characterise the life-time of an ACC: *ACC negotiation* and *ACC use*. In order to take part actively to a multi-agent organisation, to access its resources as well as to interact with its members, an agent must first *negotiate* to get an ACC from the supporting multi-agent infrastructure. In this stage, the agent typically specifies the structure (society, group) it aims at entering, and the role(s) it aims at playing. In the case of a successful negotiation – i.e. the agent request is compatible with the organisation rules –, an ACC is configured to contain all the rules applying to the requesting agent (specialised according to the specific agent request), and then released to the agent itself. From then on, the agent works as an active part of the society/group, and can *use* the ACC to act and interact according to the organisation rules defined in the ACC configuration, and enforced by the ACC at run time.

Which precise acceptations of organisation, structure, society, or group are required by the very notion of ACC, it does not come along with the definition of ACC – and this also should explain why the lines above could appear vague under that respect. *Per se*, the requirements for the ACC notion from an organisational viewpoint are mostly a notion of agent identity (and a procedure to make it known and certain), and the existence of an organisational model (if any) defining roles for agents, and associating admissible actions to roles through some organisation (security, coordination) policies. That is why the integration of the ACC notion within the RBAC-MAS framework is seamless, as formally shown in Section 4.

2.2.2 Security

When dealing with agents within a MAS, two basic stages can be devised out that frame security issues: agent entrance into a multi-agent organisation, and agent interaction within the organisation itself. The two stages can be mapped directly on the two basic stages that concern the life cycle of an ACC, i.e. ACC negotiation and use, respectively.

Security policies governing agent entrance affect ACC negotiation, through the constraints specified as agent-role and inter-role relationships. Such policies determine whether an agent can join a society, based on the agent identity – authentication is typically required here – as well as on the roles that the agent is possibly already playing within the multi-agent organisation.

Security policies governing agent interaction within an organisation are enforced at execution time by the ACC working as a run-time abstraction, ruling agent communication and access to the organisation resources according to the permissions granted to the agent role(s). Such permissions are expressed in terms of the patterns of actions and interaction protocols that an agent is allowed to perform within the organisation.

Both ACC negotiation and use stages deal with crucial security issues. However, in this paper we mostly neglect the ACC negotiation stage, since it is a quasi-static process whose complexity is not much higher than the typical authentication and role-assignment phases of any role-based system: how an agent and a specific MAS actually negotiate the initial configuration of the ACC released to the agent is not really relevant here. Instead, complexity lays in the dynamics of the agent (inter)acting within a multi-agent organisation, and specifically on how an ACC controls, constraints, filters – one may say *secures* – the agent observable behaviour within the organisation: that is why in the following we focus on the ACC use stage.

2.2.3 Coordination

In the gray area where coordination policies are not so easy to distinguish from security policies, it should be first of all noted that ACCs promote a notion of admissible agent action which is not (necessarily) statically defined. In fact, generally speaking, the actions that an agent is allowed to perform by its ACC are not assigned statically to the agent by virtue of its initial role assignment: instead, they may dynamically depend on a number of factors such as agent interaction history, time constraints, general system status and behaviour, and many others.

For instance, the capability of specifying the set of admissible actions by virtue of the agent interaction history can be used to bound the agent behaviour to some given interaction protocol, defined according to the current agent role. A well-known example in the MAS field is the Contract Net Protocol [34]: through suitably configured ACCs, it is possible to constrain the interactive behaviour of agents to follow any of the possible action sequences established for the auctioneer and bidder roles involved in the protocol. This capability is almost mandatory for MAS models and infrastructures, especially in those contexts where the enforcement of interaction protocols and social norms is strictly required, as in the case of computational institutions.

ACCs also help bridging the traditional dualism between subjective and objective coordination approaches – where coordination is interpreted as either an agent activity, or an activity over the agents, respectively [29]. In fact, an ACC on the one hand specifies and constrains the space of actions that are *objectively* available to a specific agent (role) within a MAS; on the other hand it makes such a space available to the *subjective* understanding and deliberative process of the individual agent.

For instance, in the case of a coordination infrastructure like TuCSoN, it completes the coordination tools that engineers need in order to govern the agent interaction space: laws and policies meant to manage the interaction among a group of agents are to be enforced by coordination media; instead, laws and policies meant to rule the individual actions and perceptions with respect to the ensemble are to be enforced by the ACC [25]. In other terms, an ACC is meant to be a *local* abstraction, complementing coordination media that work as *global* abstractions. In fact, an ACC contains and manages information related only to the interaction state of the specific agent owning it. Dually, a coordination medium typically manages information about the interaction and coordination state of the whole ensemble of agents that use the medium itself.

2.3 ACC Examples

Inspectability is one of the key features of abstractions handling interaction in open systems [28]. The ability of reading an ACC configuration, and understanding the admissible behaviours it allows for, is fundamental for intelligent agents which aim at entering a multi-agent organisation – first, to decide if joining the organisation could be of some interest, then to help elaborating plans of action.

From this viewpoint, which language is used to describe the ACC configuration, and make it available to the agent, is a relevant issue indeed. Making such a language be the same actually used to enforce the ACC configuration, and also match the syntax used in the formal specification of the ACC (Section 4), actually promotes conceptual integrity throughout the whole MAS engineering process (from design to deployment and execution time). The syntax for ACC configuration adopted is then not only a means to describe the behaviour of ACC, but also a language for the specification of security properties – to be exploited both at design time to a-priori shape the space of agent interaction, and at run time to enact security policies and also to make them explicitly available to agents for inspection.

So, in the following we anticipate the syntax and semantics for ACC configurations described Section 4 – and in particular the fragment for ACC policy definition, as defined in Fig. 7 on page 168 –, and exploit it to provide the readers with some simple but intuitive examples of how ACCs can be used to define and enforce a number of security policies that go beyond the naive idea of permissions (i.e., enabling a subset of actions). In fact, we show how an ACC can be used to enable protocols, enforce concurrency policies, measure and rule resource access, and allow for dynamic access control.

In order to gain the required expressive power, the language adopts features that are borrowed from well-known and deeply-studied foundational calculi for interaction, such as CCS [18] and ACP [2]. Also, in the illustration of examples below we first refer to abstract actions, then use LINDA [13] coordination primitives over tuple spaces – `in`, `rd`, and `out`[1] – as concrete instances of abstract operations. This choice on the one hand prevents us from being excessively abstract, on the other hand allow us to give evidence to some subtle issues, such as the ones related to substitutions.

2.3.1 Filtering

The simplest security policy that ACC policies should be able to express is the ability of discerning admissible agent actions, namely, which actions an agent is actually allowed to execute. This is the typical approach to security of systems based on ACLs (access control lists): once an agent has been authenticated – in our framework, once it has successfully negotiated an ACC – it has limited permissions to act on the available resources.

A general definition of the kind

$$\mathcal{D} := (a_1 + a_2 + \ldots + a_n); \mathcal{D}$$

implements the idea: the ACC policy expressed by $\mathcal{D}$ can repeatedly and indefinitely execute any of the actions $a_1, a_2, \ldots, a_n$, whereas any other action is not allowed. Hence, the choice operator $+$ is used here to enable a subset of actions, while sequential composition ; along with the recursive definition $\mathcal{D}$ makes it possible to iterate this behaviour indefinitely. In fact, any (either finite or infinite) sequence of the actions $a_1, \ldots, a_n$ is an admissible action history for an agent ruled by the ACC policy $\mathcal{D}$ above.

By using substitutions, admissible actions can be filtered in a more flexible way. In the LINDA case, for instance, we can define the policy

$$\mathcal{D} := (\texttt{out}(id, t(X,1)) + \texttt{in}(id, t(X,1)) + \texttt{rd}(id, t(X,1))); \mathcal{D}$$

[1] In particular, $\texttt{in}(id, t)$ / $\texttt{rd}(id, t)$ respectively consumes / reads a tuple matching template t from the tuple space id, whereas $\texttt{out}(id, t)$ puts tuple t in the tuple space id.

which allows only tuples of the kind $t(X, 1)$ to be actually inserted, read, and removed from tuple space id.[2]

2.3.2 Protocols

The mechanisms of ACLs is indeed a static one: the subset of the admissible actions for an agent is determined after the agent authentication, during a successful ACC negotiation, and is typically left unmodified thereafter. Instead, our language for ACC policies is expressive enough to allow for the definition of protocols of actions, that is, to impose the order in which an agent is allowed to perform a sequence of actions. For instance, by the ACC policy defined by

$$\mathcal{D} := (a_1 + a_2);\ (a_3 + a_4);\ a_5$$

the agent is only allowed to execute a sequence of three actions: first, either a_1 or a_2, then either a_3 or a_4, finally a_5. In general, by composition of the operators $+$ and ; a policy can specify as admissible any finite subsequence of actions, such as e.g.

$$\mathcal{D} := (a_1 + a_2;\ a_3) + a_1;\ a_1;\ (a_2 + a_3)$$

The same mechanism can be fruitfully exploited along with recursive definition and substitutions, as in the following LINDA case:

$$\mathcal{D} ::= \texttt{in}(id, t(X, 1));\ \texttt{out}(id, t(X, 1));\ \mathcal{D}$$

The above ACC policy on the one hand specifies that the agent should alternately consume and produce tuples of the kind $t(X, 1)$; on the other hand, by exploiting the substitution mechanism, it forces the agent to insert the same tuple just after its removal. For instance, one can easily verify that the sequence of actions $\langle \texttt{in}(id, t(1, 1)), \texttt{out}(id, t(1, 1)), \texttt{in}(id, t(2, 1)), \texttt{out}(id, t(2, 1))\rangle$ is allowed by the ACC policy above, whereas $\langle \texttt{in}(id, t(1, 1)), \texttt{out}(id, t(2, 1))\rangle$ is not. This kind of specification is particularly useful when the occurrence of a tuple in the shared space stands for the availability of a shared resource: using the above policy would ensure that agents release the resource used before accessing the others.

2.3.3 Fine-grained policies

Besides the basic mechanisms for filtering admissible actions and enforcing protocols, ACC policies allow for a finer-grained control of agent actions.

As a first example, suppose that actions consist of accesses to resources. In some cases, it might be sensible to a-priori restrain agent access to resources by limiting the number of actions allowed to the agent: the composition operator $\|$ can be exploited to this end. By the ACC policy defined as

$$\mathcal{D} ::= a_1 \| a_1 \| a_1 \| a_2 \| a_3$$

an agent can invoke a_1 at most three times, a_2 and a_3 only once – in whichever order.

[2] By the way, this example also shows why we make the substitution operator not affect definitions (see Fig. 7 on page 168): otherwise, inserting tuple $t(1, 1)$ would have caused the substitution of X with 1 from then on, preventing the subsequent insertion of e.g. tuple $t(2, 1)$.

When this mechanism is used along with protocols, it can be exploited to allow an agent to participate in more than one simultaneous conversation of the same protocol. The LINDA example shown above can be extended as

$$\mathcal{D} ::= (\texttt{in}(id, t(X, 1)); \texttt{out}(id, t(X, 1)))\|\mathcal{D}$$

allowing the agent to exploit more than one resource at the same time. In this case, on the one hand the agent is no longer required to release a resource before asking for another; on the other hand the agent still cannot insert tuples that it had not previously removed – which at least ensures safety of the coordination medium.

2.3.4 Dynamic controls and non-determinism

An interesting subtlety comes in when interpreting the meaning of the choice operator within ACC policies. Consider a policy of the kind $a_1; C_1 + \ldots + a_n; C_n$, with all the a_i different from each other. This policy allows an agent to choose which action a_i to execute; after execution, the corresponding policy continuation C_i carries on. In this case, the agent choice deterministically determines the prosecution of the ACC – and of the corresponding agent admissible interaction history as well.

Everything changes when the a_i are not all distinct. Given an ACC policy of the form $a; C + a; D$, an agent is bound to choose action a; however, after a execution, the policy may be either move to continuation C or D, non-deterministically, independently of the agent choice. An interesting interpretation of this form of non-determinism is that this sort of ACC policy allows the infrastructure to dynamically control the behaviour of an ACC at run time, so as to possibly change the policies ruling agent interaction during execution. For instance, this would allow an infrastructure to dynamically tune up access control policies based on run-time aspects such as the availability of resources, or the required quality of service.

As an example, consider the following (generic) definition of an ACC policy:

$$\mathcal{D} ::= start; (resource(r) + resource(r')); \mathcal{D}$$

Each time an agent needs to access a resource, it first executes the initialisation action $start$, then the infrastructure is allowed to determine at run time which resource between r and r' is to be made available to the agent, by suitably governing the subsequent evolution of the ACC policy $(resource(r) + resource(r'))$.

Another interesting case is related to the LINDA protocol mentioned above. The following variation of it

$$\mathcal{D} ::= \texttt{in}(id, t(X, 1)); \texttt{out}(id, t(X, 1)) + \texttt{in}(id, t(X, 1)); \texttt{out}(id, t(X, 1)); \mathcal{D}$$

makes it possible for the infrastructure to choose at any time if a given ACC has to be terminated, after any number of action pairs $\texttt{in}(id, t(X, 1)); \texttt{out}(id, t(X, 1))$ executed by the agent. In fact, each time the tuple removed by the primitive `in` is restored by the dual invocation of the primitive `out`, the policy allows the infrastructure to either choose to make it be the last agent operation (left choice), or to permit other agent actions thereafter (definition recursion, right choice).

As an alternative and equivalent formulation, the above protocol can be written as

$$\mathcal{D} ::= \texttt{in}(id, t(X, 1)); \texttt{out}(id, t(X, 1)); (\diamond + \mathcal{D})$$

where term $\diamond$ is used to denote a state where the protocol can be quit: hence, after executing the `in` and `out` operations, the protocol can be quit or keep with other new operations.

3 From RBAC to RBAC-MAS

When engineering complex software systems with agent-oriented abstractions, organisation emerges as a fundamental dimension [37], strictly related to coordination and security – as already discussed above. The connection between organisation and security is also quite apparent in RBAC (Role-Based Access Control) models / architectures, which are currently considered as the most promising approach to the engineering of security – in particular of access control – in complex information systems [33]. On the one side, RBAC major properties are the ability to articulate and enforce enterprise (system) specific security policies and to streamline the burdensome process of security management [10]: with respect to previous approaches (such as discretionary and mandatory access control), they allow for a more flexible and detailed control and management of security. On the other side, RBAC approaches make it possible to specify security policies in terms of organisation abstractions – such as roles, role permissions and inter-role relationships –, so as to easily integrate security in contexts where organisation is explicitly defined with such abstractions. Recent works have also emphasised how effective RBAC approaches are in supporting organisation and security at the infrastructure level for system heavily based on coordination: among the others, the most relevant example are Workflow Management Systems [14, 4, 1].

In [27] we showed how RBAC-like models can be suitably introduced in a MAS context, integrating a high-level role-based security approach with agent-based coordination and organisation, where the role abstraction is already at play. While MAS organisational models based on roles are typically exploited at design time, a RBAC-like model in a MAS context extends its scope to execution time: roles, sessions, policies become run-time aspects of a MAS organisation, which are dynamically manageable by means of suitable services provided by the infrastructure. So, in the same way as RBAC approaches brings at the infrastructure level security issues that were previously faced (in a similar way) by each individual applications, we aim at factoring out security issues that frequently emerge in the engineering of agent systems, and extending the MAS infrastructure with the corresponding services – suitably integrated with the MAS organisation and coordination model.

In the overall, RBAC models are gaining interest also within the MAS field for a number of good reasons. First of all, they promote conceptual integrity in MAS engineering, by framing security issues so that the basic bricks of the organisation model (roles, policies, etc) can be adopted as viable security abstractions. Also, RBAC approaches allow security policies to be defined and enforced despite the typical MAS heterogeneity and openness: on the one side, the same model can be applied in the context of systems composed by agents with heterogeneous computational models, from reactive to cognitive based ones; on the other side, the model supports the openness that is a typically desired MAS feature, in terms of dynamism of the system structure (e.g., the set of agents), and of the organisational / coordination policies as well.

In the remainder of this section, we first give an overview of RBAC models (Subsection 3.1), along with a reference architecture (Subsection 3.2), then we discuss our RBAC-MAS model (Subsection 3.3) informally, by mainly discussing its peculiarity with respect to traditional RBAC approaches. The formal discussion of the full RBAC-MAS model is instead delegated to the next section.

3.1 RBAC models overview

In RBAC, a *role* is properly viewed as a semantic construct around which access control policy is formulated, bringing together a given collection of users and permissions, in a transitory way [33]. The role concept assumes several manifestations, which RBAC aims at accommodating and capturing. A role can represent competency to do specific tasks, such as a physician or a pharmacist; but also the embodiment of authority and responsibility, such as in the case of a project supervisor. Authority and responsibility are distinct from competency.

The RBAC approach makes it possible to establish relations between roles, between permissions and roles, and between users and roles. For example, two roles can be established as mutually exclusive, so that the same user is not allowed to take on both roles at the same time. By means of inheritance relations, one role can inherit permissions assigned to a different role. These inter-role relations can be used to enforce security policies that include *separation of duties* and *delegation of authority*. Separation of duties is achieved by ensuring that mutually-exclusive roles are invoked to complete a sensitive task, such as requiring an accounting clerk and an accounting manager to participate in issuing a check.

With separation of duty, RBAC directly supports other two well-known security principles: *least privilege* and *data abstraction*. Least privilege is supported because RBAC can be configured so that only those permissions required for the tasks conducted by members of the role are assigned to the role. Data abstraction is supported by means of abstract permissions such as credit and debit for an account object, rather than the read, write, execute permissions typically provided at the operating system level. However, RBAC is also said to be *policy neutral*, since it does not enforce itself any specific access policy.

Summing up, RBAC allows for encapsulation of security policies. Access control strategy is encapsulated in various RBAC components such as role-permission, user-role and role-role relationships. Altogether, such components, which are dynamically configurable by system administrators, determine whether a particular user will be allowed to access a particular piece of data or a resource in the system. Moreover, RBAC approach makes it possible and easy to incrementally evolve the access control policy during the system life-cycle, so as to meet the ever-changing needs of a complex organisation.

3.2 The RBAC reference architecture

According to the reference architecture formally defined in [11], the main components of a RBAC model are depicted in Fig. 2, defined in terms of basic element sets and their relationships. The basic element sets are users (`USERS`), roles (`ROLES`), objects (`OBJS`), operations (`OPS`), permissions (`PERMS`) and sessions

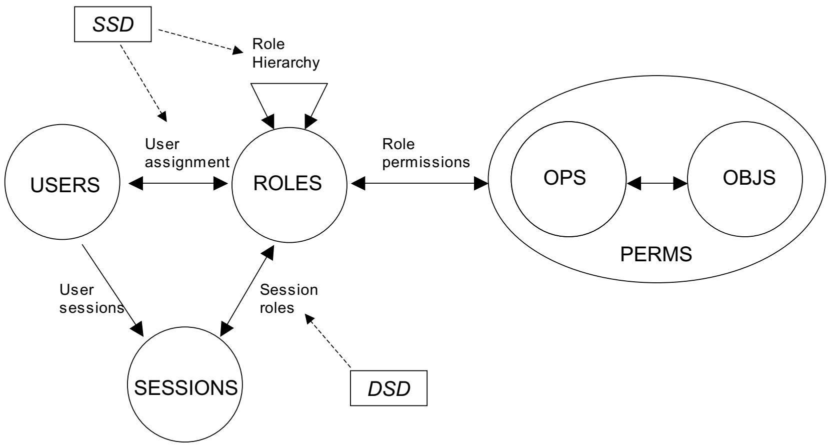

Fig. 2 RBAC Reference Model [11]

(`SESSIONS`). Users are assigned to roles and permissions to roles. A *role* is understood as a job function within the context of an organisation with some associated semantics regarding the authority and responsibilities conferred to the user assigned to the role [11]. A *permission* is an approval to perform an operation on one or more protected objects. The semantics of the term *operation* and *object* depends on the specific cases. Each session is a mapping between a user and an activated subset of roles that are assigned to the users. Each session is associated with a single user and each user is associated with one or more sessions. Hierarchies are a natural means for structuring roles so as to reflect the line of authority and responsibilities within an organisation, and define an inheritance relationship among the roles: role `R1` inherits role `R2` if all the privileges of `R2` are also privileges of `R1`.

Security policies are defined in terms of relationships between the element sets. User-assignment relationships define which users are assigned a specific role, which means that they are allowed to play it inside the organisation; permission assignment defines which permissions are assigned to each role. Static separation of duty properties (SSD) are obtained by enforcing constraints on the assignment of users to roles; instead, dynamic separation of duty properties (DSD) are obtained by placing constraints on the roles that can be activated within or across a user's sessions.

3.3 From RBAC to RBAC-MAS: the main issues

In [27] we provided the conceptual foundations to RBAC-MAS, by taking the basics of the RBAC model, and bringing them to the MAS field. The MAS perspective, in fact, introduces a new view over interaction – which are the entities interacting, and how they interact –, and so, over security, organisation and coordination. As a result, when coming to MAS, RBAC has to be suitably amended and extended to suit the nature of the agent-oriented abstractions.

Accordingly, in the rest of this section we shortly introduce RBAC-MAS by pointing out the main differences between RBAC-MAS and the original RBAC,

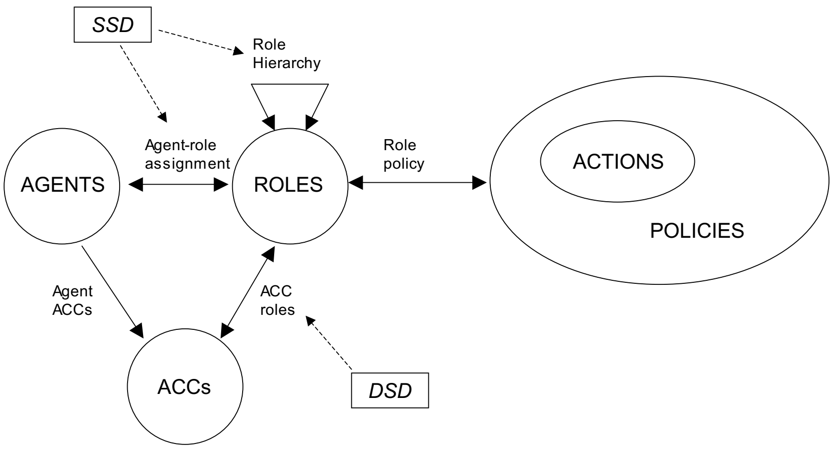

Fig. 3 A Possible Reference Model for RBAC-MAS

and leaving the formal RBAC-MAS definition to the next section. In fact, the main peculiarities of RBAC-MAS with respect to RBAC, as they emerge by comparing Fig. 2 with Fig. 3, can be summarised as follows:

Agent Classes – Instead of RBAC *users*, RBAC-MAS has *agent classes*, each one characterised by a unique identifier. To each identifier, a cardinality (even infinite) is associated, stating how many agents at most can belong to the class – which gives RBAC-MAS a finer control over system *openness*.

Actions – Operations and objects are not unrelated in MAS. Therefore RBAC-MAS introduces a notion of *action*, which is an *operation* over a given *object*. Actions are terms with variables, and a notion of *substitution* is introduced, mainly to cope with issues like action completion, perceptive actions, and so on.

Policies – Instead of *permissions*, which are subsets of the cartesian product of the set of the operations and the set of objects, RBAC-MAS introduces *policies*, which are protocols (possibly infinite, concurrent, and non-deterministic) of actions – as illustrated by the examples in Subsection 2.3. Policies are first-class entities, with their own unique identifier.

ACCs and Sessions – In the RBAC-MAS approach, the ACC abstraction also accounts for the RBAC notion of session. Since each agent is assigned its own ACC within an organisation, RBAC-MAS allows only one session per agent (per organisation). Getting rid of an ACC and asking for a new one corresponds then to an agent closing a session and trying to open a new one.

A number of further features of RBAC-MAS are worth to be pointed out:

Static / Dynamic Enforcement of Policies – In the RBAC-MAS approach, policies are dynamically enforced via ACCs, which work as both design time and run time abstractions. Correspondingly, no conceptual distinction can be made any longer between SSD and DSD. Also, given that the same abstraction that mediates all the agent interactions (ACC) is used to design, contain and enforce security policies, any agent is structurally prevented from executing prohibited actions within an organisation.

Role Activation / Deactivation – Policies constrain which roles can be activated or deactivated by an agent given its current state of actions and active rolesets. So, for instance, policies can be specified not only to ensure a given path of actions for an agent, but also to enforce an allowed sequence of role activation and deactivation. An ACC can be left only when all its roles have been deactivated.

No Default Roleset – RBAC-MAS does not provide a notion of default roleset to an agent. Instead, the agent initially enters a void ACC, and can subsequently activate roles on a step-by-step basis, according to the role activation/deactivation policies it has been assigned.

Finally, in order to keep the RBAC-MAS presentation simpler, and make the formalism more manageable and understandable by the reader, in this paper we stick to the simpler case of one organisation per MAS – the more general case where a MAS contains more than one separate organisation structure would bring nothing new from a conceptual viewpoint here. For the same reason, we neglect in the following the issue of role hierarchies – merely for the fact that RBAC-MAS adds nothing to this RBAC notion.

4 The algebraic approach

In this section, we formalise our conceptual framework, by using an algebraic approach to denote MASs and their evolution. In Subsection 4.1, we first introduce some basic notation, then we implicitly define our ontology by formally representing the essential ingredients of RBAC-MAS – an agent, an organisation, a MAS, an agent action, an ACC – as well as their mutual relationships. Subsequently, in Subsection 4.2, the semantics of RBAC-MAS is formalised through a number of transition rules defining the admissible evolutions of a RBAC-MAS system.

4.1 Notation, syntax & ontology

As a general notation convention, in the following we let Greek letters α, β, ... identify meta-variables over actions, ranging over sets denoted by notation $Set(\alpha)$, $Set(\beta)$, and so on. Non-capitalised letters a, b, ... represent meta-variables over identifiers of different kinds, ranging over sets denoted by notation $Set(a)$, $Set(b)$, and so on. In particular, n ranges over the set of natural numbers and infinite ($n \in \mathcal{N} \cup \{\infty\}$). Capitalised letters $A, B, \ldots$ are instead used as meta-variables for system components and abstractions of various sorts, ranging over sets denoted by symbols $\mathbb{A}, \mathbb{B}, \ldots$. Accordingly, the basic items of our framework are denoted as follows:

- c ranges over the set of the *agent classes*,
- a ranges over the set of the *agent identifiers*,
- r ranges over the set of the *role identifiers*,
- p ranges over the set of the *policy identifiers*,
- ϵ ranges over the set of the *environment actions*.

The sets $Set(c)$, $Set(a)$, $Set(r)$, $Set(p)$ and $Set(\epsilon)$ ranged over by the above variables are specific to any given MAS organisation: so, no further hypothesis are made on their structure.

If we denote the union of the above sets with $\mathcal{T}$, ranged over by variable τ, a notion of *substitution* can be introduced for terms τ. We name substitution a total function $\sigma \in \Sigma$ from terms to terms, where $\Sigma \subseteq \mathcal{T} \mapsto \mathcal{T}$. Notation $\sigma\tau$ denotes a substitution σ applied to a term τ, and symbol $\circ$ is used for composition of substitutions – which is assumed to be a closed operator over Σ. As usual, this notation is often abused by allowing a substitution to be applied to elements of any syntactic category: the meaning is that any term τ occurring inside the elements gets correspondingly substituted. For instance, if we consider the expression term $\tau \| \tau'; \tau''$, we let $\sigma(\tau \| \tau'; \tau'')$ be syntactically equal to $(\sigma\tau) \| (\sigma\tau'); (\sigma\tau'')$.

It is worth noticing that substitutions introduce a (pre)order relation over terms – a relation which is reflexive and transitive. We say that τ is more specific than τ' – written as $\tau \preceq \tau'$ – iff there exists at least one substitution $\sigma \in \Sigma$ such that $\sigma\tau = \tau'$.

For any of the above meta-variables, say it is x, we let $\widetilde{x}$ to range over bound values, that is, values such that for any substitution σ we have $\sigma\widetilde{x} = \widetilde{x}$.

4.1.1 MAS configuration

At any given time a MAS is denoted by its *configuration*, which captures an instant of the MAS evolution over time. As summarised by Fig. 4, a MAS is conceived here as composed by three parts – *agents*, *organisation*, and *environment*. Correspondingly, in our framework, a MAS configuration is composed by three sets:

Agent configuration — X is the *agent configuration* of a MAS, that is, at any given time, the set of the agents currently belonging to the MAS. Every agent $x \in X$ is either an *active* or an *inert* agent. An active agent is an agent that has already entered the organisation – so, it has an ACC assigned yet –, which is denoted by a term $\langle a, (C)A \rangle$, where a is the agent identifier, A the *agent action configuration* and C its *ACC configuration*. An inert agent is an agent that has not yet

S	$::=$	$X \| O \| E$	MAS configuration
X	$::=$	0	agent configuration
	$\mid$	$\langle a, A \rangle$	inert agent
	$\mid$	$\langle a, (C)A \rangle$	active agent
	$\mid$	$X \| X$	agents
O	$::=$	$K \| V$	organisation configuration
K	$::=$	0	structure configuration
	$\mid$	$\{c(n)\}_{\mathbb{C}}$	agent class (with cardinality)
	$\mid$	$\{c, r\}_{\mathbb{CR}}$	class role
	$\mid$	$\{r, p\}_{\mathbb{RP}}$	role policy
	$\mid$	$\{p := P\}_{\mathbb{P}}$	policy definition
	$\mid$	$\{[RS] + r\}_{\mathbb{SD}}$	separation of duty constraint
	$\mid$	$K \| K$	control structures
RS	$::=$	$0 \mid r \mid RS \| RS$	roleset
V	$::=$	0	activity configuration
	$\mid$	$[a, c]_{\mathbb{A}}$	active agent class
	$\mid$	$[a, r]_{\mathbb{AR}}$	active agent role
	$\mid$	$V \| V$	activity controls

Fig. 4 MAS configuration

entered the organisation – so, it has no ACC assigned yet –, which is denoted by a term $\langle a, A\rangle$.

Organisation configuration — O is the *organisation configuration* of a MAS, that is, the set of the organisation structures that shape the MAS at any given time. In turn, organisation configuration is conceptually divided in two parts. The *structure configuration* K includes information on the *control structures* that define the organisation, that is, recognised agent classes, roles, permissions, separation of duty constraints, and their relationships. In particular, $\{c(n)\}_{\mathbb{C}}$ means the organisation recognises agent class c with cardinality n, $\{c, r\}_{\mathbb{CR}}$ that role identifier r is associated to agent class c, $\{r, R\}_{\mathbb{RP}}$ that R is the role specification associated to r, $\{p := P\}_{\mathbb{P}}$ that P is the policy specification associated to identifier p, and finally $\{[RS] + r\}_{\mathbb{SD}}$ is the separation of duty rule saying that an agent having activated the roleset RS cannot activate role r. The *activity configuration* V instead keeps track of the status of the controls enforced on agents, namely, the class c of each active agent a by $[a, c]_{\mathbb{A}}$, and by $[c, r]_{\mathbb{AR}}$ that an agent with class c is playing role r.

Environment configuration — E is the *environment configuration* of a MAS, that is the set of the elements – such as information sources, physical devices, artifacts of various kinds [26] – which altogether constitute the MAS environment. Environment configurations $E \in \mathbb{E}$ will not be described in their syntax and semantics in the following, since they are peculiar to any specific MAS, so not easily generalised.

4.1.2 Agent behaviour & actions

As it deals with the control of interactions between the components of a system, RBAC is not concerned with the internal structure of the components, but only with their interactive behaviour. Analogously, our framework makes no hypothesis on the agent architecture and inner dynamics, and focuses instead on modelling the agent observable behaviour. As a consequence, an agent evolution over time is represented here through its *agent action configuration* A, expressed in terms of the agent actions – actions on the environment, the control structures and the ACC, as reported in Fig. 5 and pictorially summarised in Fig. 6.

More precisely, variable ϵ (as previously discussed) ranges over actions on the environment – upon which we do not make any hypothesis. Variable ω ranges over actions on the organisation, which are used to read or adapt static control structures – adding, removing, and updating users, roles, policies, separation of duty constraints. Variable ϕ ranges over actions on either the environment or the organisation. Finally, variable ν ranges over actions handling ACC and its structure, including entering and leaving the ACC, as well as activating and deactivating a role.

Actions ϕ over the environment and the organisation by an agent are the ones directly perceived by the rest of the MAS: as such, they are actually enabled and controlled by the ACC according to the roles the agent is playing and its associated policies, hence an agent executes them by qualifying role and policy identifier $(r : p : \phi)$. Instead, ACC actions ν have no direct effect on the MAS, but should also be handled and recorded globally at the infrastructure level: correspondingly, ι represents the actions for ACC negotiation or change as perceived by the rest of the MAS, that is, an action ν qualified by the agent identifier a.

$A ::= 0 \mid \alpha.A \mid A + A$	agent action configuration
$\alpha ::= r : p : \phi \mid \nu$	agent actions
$\phi ::= \epsilon \mid \omega$	environment / organisation actions
$\omega ::=$	organisation actions
$+K$	control structure addition
$\mid -K$	control structure removal
$\mid K \mapsto K$	control structure update
$\mid ?K$	control structure reading
$\nu ::=$	ACC actions
$\downarrow$	ACC entry
$\mid \uparrow$	ACC exit
$\mid +r : R$	role activation
$\mid -r$	role deactivation
$\iota ::= a\nu$	negotiation actions

Fig. 5 Agent behaviour & actions

4.1.3 ACC configuration & policies

An ACC enforces and controls agent actions within an organisation. Every ACC is associated with an active agent, and its state, called *ACC configuration*, evolves along with the evolution of the agent behaviour. Re-reading notation in Fig. 4, an active agent $\langle a, (C)A \rangle$ has agent identifier a, agent action configuration A and ACC configuration C.

According to Fig. 7, ACC configurations are a parallel composition of roles R – for an agent concurrently playing more than one role – each specified by its own role identifier r. Since it is possibly characterised by more than one policy, a role is itself a composition of policies P, each with its own policy identifier p. As a

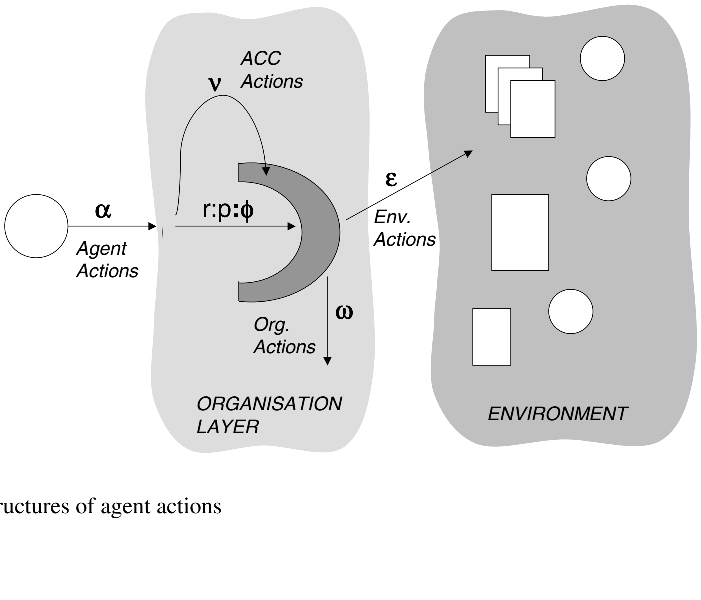

Fig. 6 Structures of agent actions

C	$::= 0$	ACC configuration
	$\mid\ r : R$	active role
	$\mid\ C \| C$	ACCs
R	$::= 0$	role
	$\mid\ p : P$	active policy
	$\mid\ R \| R$	roles
P	$::= 0$	policy
	$\mid\ \pi$	controlled action
	$\mid\ P ; P$	sequence
	$\mid\ P + P$	choice
	$\mid\ P \| P$	interleaving
	$\mid\ \mathcal{D}$	(recursive) definition call
π	$::= \phi \mid \diamond$	policy actions

Fig. 7 ACC configuration

result, an active policy is identified by qualifying the role identifier and the policy identifier ($r : p : P$).

ACCs control actions π through policies (policy actions in Fig. 7), expressing constraints on the occurrences of actions ϕ, as well as of the *escape action* $\diamond$, used to leave a policy. Such constraints are imposed through the operators for sequential composition ";", choice "+", parallel composition "||", and (recursive) definitions "$\mathcal{D}$".

4.2 Operational semantics

In the following, we provide a specification of RBAC-MAS in terms of the admissible evolutions of a MAS configuration, modelled in terms of transition systems over the process algebraic approach described above.

4.2.1 Agent behaviour & actions

The semantics of the agent behaviour is modelled by a transition system $\langle \mathbb{A}, \longrightarrow_{\mathbb{A}}, Set(\alpha)\rangle$ denoting the evolution over time of agent action configurations. As shown in the formal syntax, we adopt a process algebraic approach in the definition of A, modelling continuation and non-deterministic choice. As usual, we introduce the following congruence relation, stating basic properties of choice

$$A + 0 \equiv A \qquad A + A' \equiv A' + A \qquad (A + A') + A'' \equiv A + (A' + A'')$$

and define the operational rules:

$$\frac{A \xrightarrow{\alpha}_{\mathbb{A}} A''}{A + A' \xrightarrow{\alpha}_{\mathbb{A}} A''}\ \text{[A-CHO]}$$

$$\frac{-}{\alpha.A \xrightarrow{\sigma\alpha}_{\mathbb{A}} \sigma A}\ \text{[A-SEQ]}$$

Rule [A-CHO] provides the usual semantics to the choice operator: as an alternative is selected for executing an action, the other choice is discarded. Rule [A-SEQ] instead deals with action execution and continuation: as the agent intends to execute action α, the more specific version $\sigma\alpha$ can actually be executed, in which case substitution σ is applied to the continuation A.

4.2.2 Policy

A main difference between RBAC and RBAC-MAS is grounded in the notion of policy, which actually differs from that of permission. While permissions simply define a set of allowed operations over objects, we rely here on the stronger notion of protocol: a policy is an admissible protocol of actions over environment and organisation, possibility featuring non-determinism, interleaved sub-policies, and recursive behaviours. Moreover, at any given time a policy could allow for a more specific action than it is actually prescribed; furthermore, the environment which the action is executed over could constrain the action to be even more specific. We tackle this aspect by let substitutions make into actions. As the actions controlled / enforced by a policy are ranged over by π, we define

$$\overline{\pi} ::= \sigma \rhd \pi$$

An *interaction* $\sigma \rhd \pi$ means that a further substitution σ is actually externally imposed to the allowed action π. Therefore, the transition system for policies is defined by the triple $\langle \mathbb{P}, \longrightarrow_{\mathbb{P}}, Set(\overline{\pi})\rangle$. Similarly to the case of agent action configurations, we introduce the congruence rules

$$\begin{gathered}
0 + P \equiv P \qquad P + P' \equiv P' + P \qquad (P + P') + P'' \equiv P + (P' + P'') \\
0 \| P \equiv P \qquad P \| P' \equiv P' \| P \qquad (P \| P') \| P'' \equiv P \| (P' \| P'') \\
0;\, P \equiv P;\, 0 \equiv P \qquad P;\, (P';\, P'') \equiv (P;\, P');\, P'' \\
(P + P');\, P'' \equiv (P;\, P'') + (P';\, P'')
\end{gathered}$$

and the operational rules:

$$\frac{-}{\pi;\, P \xrightarrow{\sigma' \rhd \sigma\pi}_{\mathbb{P}} \sigma'\sigma P} \qquad \text{[P-ACT]}$$

$$\frac{\mathcal{D} := P \qquad P \xrightarrow{\overline{\pi}}_{\mathbb{P}} P'}{\mathcal{D} \xrightarrow{\overline{\pi}}_{\mathbb{P}} P'} \qquad \text{[P-DEF]}$$

$$\frac{P \xrightarrow{\sigma \rhd \pi}_{\mathbb{P}} P'}{(P \| P'');\, P''' \xrightarrow{\sigma \rhd \pi}_{\mathbb{P}} (P' \| \sigma P'');\, \sigma P'''} \qquad \text{[P-PAR]}$$

$$\frac{P \xrightarrow{\overline{\pi}}_{\mathbb{P}} P'}{P + P'' \xrightarrow{\overline{\pi}}_{\mathbb{P}} P'} \qquad \text{[P-CHO]}$$

Rule [P-ACT] defines the crucial aspect of policy interactions; as action π is specified in the policy, any more specific version $\sigma\pi$ can be executed which can be further constrained by external imposition of substitution σ' (by the environment): in this case, $\sigma \circ \sigma'$ is actually applied to the policy continuation. Rule [P-DEF] handles (recursive) definitions; a definition symbol $\mathcal{D}$ behaves as the policy P it is associated with. Rule [P-PAR] defines the semantics of parallel composition: if policy P, in parallel with P'' and followed by P''', allows for action π and substitution σ, then this substitution is to be applied both to P'' and P'''. Rule [P-CHO] is the standard rule for the choice operator.

4.2.3 Agent coordination context

The ACC and its relationship with an agent is expressed by the two kinds of admissible agent configurations X, which can be either of the kind $\langle a, A\rangle$ (inert agent) or $\langle a, (C)A\rangle$ (active agent) according to Fig. 4. By controlling / enforcing actions of an agent through its ACC, an agent configuration X evolves by two kinds of actions: *(i)* by an action ϕ of an agent over either the environment or the organisation, which is amenable of substitutions (hence we define $\overline{\phi} ::= \sigma \rhd \phi$), or *(ii)* by a negotiation action ι. Accordingly, the transition system that models the evolution of agent configurations is then of the kind $\langle \mathbb{X}, \longrightarrow_{\mathbb{X}}, Set(\overline{\phi}) \cup Set(\iota)\rangle$.

In particular, the operational rule for actions ϕ is as follows:

$$\frac{A \xrightarrow{r:p:\sigma\phi}_{\mathbb{A}} A' \qquad P \xrightarrow{\sigma\rhd\phi}_{\mathbb{P}} P'}{\langle a, (C\|r : (R\|p : P))A\rangle \xrightarrow{\sigma\rhd\phi}_{\mathbb{X}} \langle a, (C\|r : (R\|p : P'))A'\rangle}\ [\text{ACT}]$$

An agent playing a role r can perform an action ϕ (or a more specific one) if there exists a corresponding policy p that allows for it. Also, action execution may require action specialisation by substitution (σ in the rule above), which is to be applied to both policy P and agent action configuration A, and makes them evolve to a new state.

Substitutions of this kind are actually sufficiently expressive to model a number of narrowing / widening scenarios. Suppose that actions ϕ are terms of the kind $f(t_1, .., t_n)$, where elements t_i are either variables v or integer values n, and that the selected policy specifies action $f(v, 1)$. As a first scenario, assume the current action α in the agent action configuration is of the kind $f(v', v'')$; the ACC allows the more specific action $\phi = f(v, 1)$ – the environment could either accept it, or accept the more specific version f(2,1) by substitution $\sigma = v \mapsto 2$. As a second scenario, assume the current action α in the agent configuration is of the kind $f(2, 1)$; the ACC allows the action $\phi = f(2, 1)$ – the environment directly accepts it with identity substitution.

Operational rules dealing with negotiation actions are somehow more articulated. First of all, one should notice that the ACC does not exert any actual control over negotiation actions, since they are always admissible and potentially executable. This derives from the fact that such actions are not directly observable, since they affect the ACC configuration, and have no direct effect on the agent environment. However, since negotiation actions are exposed in the transition relation $\rightarrow_{\mathbb{X}}$, they execution can actually be controlled by the outside – by the infrastructure in particular: in fact, as shown in the following, the organisation controls them in order to enforce the required access control policies.

$$\frac{A \xrightarrow{\downarrow}_{\mathbb{A}} A'}{\langle a, A\rangle \xrightarrow{a\downarrow}_{\mathbb{X}} \langle a, (0)A'\rangle} \quad \text{[ENTER]}$$

$$\frac{A \xrightarrow{\uparrow}_{\mathbb{A}} A'}{\langle a, (0)A\rangle \xrightarrow{a\uparrow}_{\mathbb{X}} \langle a, A'\rangle} \quad \text{[QUIT]}$$

$$\frac{A \xrightarrow{+r:R}_{\mathbb{A}} A'}{\langle a, (C)A\rangle \xrightarrow{a+r:R}_{\mathbb{X}} \langle a, (C\|r : R)A'\rangle} \quad \text{[ACTIVATE]}$$

$$\frac{\langle a, (C\|r : R)A\rangle \xrightarrow{a-r}_{\mathbb{X}} \langle a, (C)A'\rangle \qquad \big(P \xrightarrow{\sigma\rhd\diamond}_{\mathbb{P}} P' \quad \text{or} \quad P \not\rightarrow_{\mathbb{P}}\big)}{\langle a, (C\|r : (R\|p : P))A\rangle \xrightarrow{a-r}_{\mathbb{X}} \langle a, (C)A'\rangle} \quad \text{[DEACT-REC]}$$

$$\frac{A \xrightarrow{-r}_{\mathbb{A}} A'}{\langle a, (C\|r : 0)A\rangle \xrightarrow{a-r}_{\mathbb{X}} \langle a, (C)A'\rangle} \quad \text{[DEACT-FIX]}$$

Rule [ENTER] initially provides an agent entering an ACC with the void context, while rule [QUIT] is used to let an agent in a void context to leave the ACC – and the organisation as well. Rule [ACTIVATE] makes an agent activate a role r with specification R – which is actually imposed/checked by the organisation –; activation simply adds to the context. Finally, rules [DEACT-REC] and [DEACT-FIX] handle the recursive deactivation of a role, policy by policy. A policy can be removed whenever it allows for the escape action $\diamond$ or it reaches a deadlock state (i.e., it is terminated); when no more policies occur for the role, this can be removed from the context.

4.2.4 MAS operational semantics

Once the evolution over time of the agent and of the organisation components have been operationally defined above, it is now possible to formally describe the dynamics of a MAS as a whole in our framework. The global operational semantics of a MAS is specified via an unlabelled transition system $\langle \mathbb{S}, \xrightarrow{S}, \{\}\rangle$, where MAS configurations evolve by the execution of three different actions, according to the following operational rules:

$$\frac{X \xrightarrow{\sigma\rhd\epsilon}_{\mathbb{X}} X' \qquad E \xrightarrow{\epsilon\rhd\sigma}_{\mathbb{E}} E'}{X\|E\|S \rightarrow_{\mathbb{S}} X'\|E'\|S} \quad \text{[S-E]}$$

$$\frac{X \xrightarrow{\sigma\rhd\omega}_{\mathbb{X}} X' \qquad O \xrightarrow{\omega\rhd\sigma}_{\mathbb{O}} O'}{X\|O\|S \rightarrow_{\mathbb{S}} X'\|O'\|S} \quad \text{[S-O]}$$

$$\frac{X \xrightarrow{\widetilde{\iota}}_{\mathbb{X}} X' \qquad O \xrightarrow{\widetilde{\iota}}_{\mathbb{O}} O'}{X\|O\|S \rightarrow_{\mathbb{S}} X'\|O'\|S} \quad \text{[S-N]}$$

Rule [S-E] handles the interaction of an agent with the environment. This occurs if one of the components of the environment (E) can execute the requested action ϵ.

As a result of action execution, such a component can actually enforce a substitution σ ($\epsilon \rhd \sigma$) which is then applied to the agent configuration as previously seen. As discussed above, we do not elaborate further upon a model for the environment, as this is out of the scope of the RBAC-MAS model – correspondingly, a transition system over E is not defined here. As shown in [26], yet, the schema action-substitution adopted is sufficiently expressive to account for bi-directional communication with other agents, as well as for interaction with artifacts of various sorts. Rule [S-O] similarly deals with organisation actions, used to access and manage the organisation control structures. If organisation action ω is admissible, the organisation could be in need (for instance, to ground the agent request) to enforce a substitution σ ($\omega \rhd \sigma$), which is then applied to the agent configuration. Finally, rule [S-N] describes the execution of negotiations actions $\widetilde{\iota}$, which affects the organisation as shown in the following.

4.2.5 Organisation

In order to complete the specification of the RBAC-MAS formal model, it is now sufficient to describe the organisational aspects – that is, how the organisation handles negotiation and organisation management actions – in terms of a transition system over O.

Negotiation For the transition system $\langle \mathbb{O}, \rightarrow_{\mathbb{O}}, Set(\iota)\rangle$, the rules dealing with negotiation actions av are as follows:

$$\frac{[a, c]_{\mathbb{A}} \notin O}{O \| \{c(n)\}_{\mathbb{C}} \xrightarrow{a\downarrow}_{\mathbb{O}} O \| \{c(n-1)\}_{\mathbb{C}} \| [a, c]_{\mathbb{A}}} \quad \text{[O-ENTER]}$$

$$\frac{-}{O \| \{c(n)\}_{\mathbb{C}} \| [a, c]_{\mathbb{A}} \xrightarrow{a\uparrow}_{\mathbb{O}} O \| \{c(n+1)\}_{\mathbb{C}}} \quad \text{[O-LEAVE]}$$

$$\frac{O \| \xrightarrow{a+r}_{\mathbb{O}} O' \| [a, r \mapsto R]_{\mathbb{AP}}}{\begin{array}{c} O \| \{r, p\}_{\mathbb{RP}} \| \{p := P\}_{\mathbb{P}} \xrightarrow{a+r}_{\mathbb{O}} \\ O' \| \{r, p\}_{\mathbb{RP}} \| \{p := P\}_{\mathbb{P}} \| [a, r \mapsto R \| p : P]_{\mathbb{AP}} \end{array}} \quad \text{[O-ACT-REC]}$$

$$\frac{\begin{array}{c} r \neq r' \qquad \{r, p\}_{\mathbb{RP}} \notin O \\ O \| \{[RS] + r\}_{\mathbb{SD}} \xrightarrow{a+r}_{\mathbb{O}} O' \| \{[RS] + r\}_{\mathbb{SD}} \end{array}}{\begin{array}{c} O \| [a, r']_{\mathbb{AR}} \| \{[RS \| r'] + r\}_{\mathbb{SD}} \xrightarrow{a+r}_{\mathbb{O}} \\ O' \| [a, r']_{\mathbb{AR}} \| \{[RS \| r'] + r\}_{\mathbb{SD}} \end{array}} \quad \text{[O-ACT-SD]}$$

$$\frac{\begin{array}{cc} [a, r]_{\mathbb{AR}} \notin O & [a, r']_{\mathbb{AR}} \notin O \\ r \neq r' & \{r, p\}_{\mathbb{RP}} \notin O \end{array}}{\begin{array}{c} O \| \{c, r\}_{\mathbb{CR}} \xrightarrow{a+r}_{\mathbb{O}} \\ O \| \{c, r\}_{\mathbb{CR}} \| [a, r]_{\mathbb{AR}} \| [a, r \mapsto 0]_{\mathbb{AP}} \end{array}} \quad \text{[O-ACT-FIX]}$$

$$\frac{-}{O \| [a, r]_{\mathbb{AR}} \xrightarrow{a-r}_{\mathbb{O}} O} \quad \text{[O-DEACT]}$$

Rule [O-ENTER] states that an agent is allowed to enter an ACC only if the organisation recognises its class c and the cardinality constraint is satisfied – namely, $\{c(n)\}_{\mathbb{C}}$ occurs in the organisation configuration (with $n > 0$, which means that another agent of class c can enter the organisation). Correspondingly, the term $[a, c]_{\mathbb{A}}$ is added to the organisation configuration and the cardinality is correspondingly decreased, so as to track the agent presence within the organisation. Dually, rule [O-LEAVE] handles agent quitting the ACC, by removing the term $[a, c]_{\mathbb{A}}$ and correspondingly increasing cardinality.

The three rules [O-ACT-*] describe role activation, which is obtained through two consecutive recursions. The first ([O-ACT-REC]) is used to construct the context-role R to be activated, defined by term $[a, r \mapsto R]_{\mathbb{AP}}$. The second ([O-ACT-SD]) checks whether separation of duty is satisfied, namely, if correspondingly to the activated roleset RS (or any of its subsets) there is a SD rule $\{[RS] + r\}_{\mathbb{SD}}$. The fixpoint rule ([O-ACT-FIX]) additionally checks if the agent is actually allowed to play that role (by occurrence of $\{c, r\}_{\mathbb{CR}}$) and if that role is not already activated for him (by absence of $[a, r]_{\mathbb{AR}}$). Finally, rule [O-DEACT] deals with agent deactivation, which is simply obtained by removing term $[a, r]_{\mathbb{AR}}$.

Organisation management As a first approximation, the operational rules dealing with organisation actions ω could be easily described as follows:

$$\frac{K \notin O}{O \xrightarrow{+K}_{\mathbb{O}} O \| K} \quad \text{[ADD]}$$

$$\frac{-}{O \| K \xrightarrow{-K}_{\mathbb{O}} O} \quad \text{[REM]}$$

$$\frac{K' \notin O}{O \| K \xrightarrow{K \mapsto K'}_{\mathbb{O}} O \| K'} \quad \text{[UPD]}$$

$$\frac{-}{O \| \sigma K \xrightarrow{\sigma \rhd ?K}_{\mathbb{O}} O \| \sigma K} \quad \text{[RD]}$$

These rules are respectively used to add, remove, update, and read a structure configuration K from the organisation. However, such rules do not deal with consistency: for instance, a policy definition could be removed while an agent is still allowed to use it, thus possibly leading the MAS configuration to get stuck when the agent activates a new role using that policy. Whenever these constraints are of some concern, one has to rely on the following refinement of the above rules:

Agent class

$$\frac{\{c(n')\}_{\mathbb{C}} \notin O}{O \xrightarrow{+\{c(n)\}_{\mathbb{C}}}_{\mathbb{O}} O \| \{c(n)\}_{\mathbb{C}}} \quad \text{[O-ADD-C]}$$

$$\frac{\{c, r\}_{\mathbb{CR}}, [a, c]_{\mathbb{A}} \notin O}{O \| \{c(n)\}_{\mathbb{C}} \xrightarrow{-\{c(n)\}_{\mathbb{C}}}_{\mathbb{O}} O} \quad \text{[O-REM-C]}$$

$$\frac{-}{O \| \{c(n)\}_{\mathbb{C}} \xrightarrow{\{c(n)\}_{\mathbb{C}} \mapsto \{c(n')\}_{\mathbb{C}}}_{\mathbb{O}} O \| \{c(n')\}_{\mathbb{C}}} \quad \text{[O-UPD-C]}$$

Rule [O-ADD-C] adds (the structure for) an agent class $\{c(n)\}_{\mathbb{C}}$ provided another one for c is not already occurring. Rule [O-REM-C] removes (the structure for) an agent class $\{c(n)\}_{\mathbb{C}}$ provided no role is assigned to c ($\{c, r\}_{\mathbb{CR}} \notin O$), and no agent of that class is currently active ($[a, c]_{\mathbb{A}} \notin O$). Finally, rule [O-UPD-C] allows for updating the structure of an agent class only by changing its cardinality (from n to n').

Class-role

$$\frac{\{c, r\}_{\mathbb{CR}} \notin O}{O \xrightarrow{+\{c,r\}_{\mathbb{CR}}}_{\mathbb{O}} O \| \{c, r\}_{\mathbb{CR}}} \quad \text{[O-ADD-CR]}$$

$$\frac{\{r, p\}_{\mathbb{RP}}, \{[RS\|r] + r'\}_{\mathbb{SD}}, \{[RS] + r\}_{\mathbb{SD}}, [a, r]_{\mathbb{AR}} \notin O}{O \| \{c, r\}_{\mathbb{CR}} \xrightarrow{-\{c,r\}_{\mathbb{CR}}}_{\mathbb{O}} O} \quad \text{[O-REM-CR]}$$

Rule [O-ADD-CR] is used to add a class role $\{c, r\}_{\mathbb{CR}}$, which is allowed only if it is not already occurring. Then, by rule [O-REM-CR], the class role can be removed only if the role has no associated policy ($\{r, p\}_{\mathbb{RP}} \notin O$), if it involves no separation of duty rule ($\{[RS\|r] + r'\}_{\mathbb{SD}}, \{[RS] + r\}_{\mathbb{SD}} \notin O$), and if no agent has currently activated that role ($[a, r]_{\mathbb{AR}} \notin O$).

Role-policy

$$\frac{\{r, p\}_{\mathbb{RP}} \notin O}{O \xrightarrow{+\{r,p\}_{\mathbb{RP}}}_{\mathbb{O}} \{r, p\}_{\mathbb{RP}}} \quad \text{[O-ADD-RP]}$$

$$\frac{[a, r]_{\mathbb{AR}} \notin O}{O \| \{r, p\}_{\mathbb{RP}} \xrightarrow{-\{r,p\}_{\mathbb{RP}}}_{\mathbb{O}} O} \quad \text{[O-REM-RP]}$$

Similarly to class roles, by rules [O-ADD-RP] and [O-REM-RP] a role policy can be added only if not already occurring, and can be removed if no agent has currently activated that role.

Policy-definition

$$\frac{\{p := P'\}_{\mathbb{P}} \notin O}{O \xrightarrow{+\{p:=P\}_{\mathbb{P}}}_{\mathbb{O}} O \| \{p := P\}_{\mathbb{P}}} \quad \text{[O-ADD-P]}$$

$$\frac{\{r, p\}_{\mathbb{RP}} \notin O}{O \| \{p := P\}_{\mathbb{P}} \xrightarrow{-\{p:=P\}_{\mathbb{P}}}_{\mathbb{O}} O} \quad \text{[O-REM-P]}$$

$$\frac{-}{O \| \{p := P\}_{\mathbb{P}} \xrightarrow{\{p:=P\}_{\mathbb{P}} \mapsto \{p:=P'\}_{\mathbb{P}}}_{\mathbb{O}} O \| \{p := P'\}_{\mathbb{P}}} \quad \text{[O-UPD-P]}$$

By rule [O-ADD-P] a policy definition can be added if not already occurring, as usual. By rule [O-REM-P], a policy definition can be removed if it is assigned to no role. Finally, rule [O-UPD-P] allows for directly updating a policy definition, changing the defined policy from P to P'.

Separation of duty

$$\frac{\{[RS]+r\}_{\mathbb{SD}} \notin O \qquad \{c,r\}_{\mathbb{CR}} \notin O \qquad (r' \in RS{:}\{c,r'\}_{\mathbb{CR}} \notin O) \qquad r \notin RS}{O \xrightarrow{+\{[RS]+r\}_{\mathbb{SD}}}_{\mathbb{O}} O \| \{[RS]+r\}_{\mathbb{SD}}} \text{[O-ADD-SD]}$$

$$\frac{-}{O \| \{[RS]+r\}_{\mathbb{SD}} \xrightarrow{-\{[RS]+r\}_{\mathbb{SD}}}_{\mathbb{O}} O} \text{[O-REM-SD]}$$

A new separation of duty rule $\{[RS]+r\}_{\mathbb{SD}}$ can be added by [O-ADD-SD] if it is not already occurring, and if the roles it specifies are assigned to some class. Finally, rule [O-REM-SD] allows for removing any separation of duty rule.

5 Related works and conclusion

In this paper, we defined a uniform conceptual framework for coordination, security and organisation aspects in MAS, and formalised it through a process-algebraic approach. Such a framework, called RBAC-MAS, is rooted in the role-based access control approaches, and exploits the notion of ACC (agent coordination context) as its basic brick. In particular, the ACC is an organisational abstraction where standard process algebra techniques are used to define a language for both specifying and enacting security / coordination policies within MAS. As discussed in [27], adopting an RBAC-like approach makes it possible to gain all the benefits of RBAC in engineering security inside complex MAS organisations, mainly in terms of encapsulation of the security policies, and flexibility in their management. There, the notion of ACC is the key for integrating coordination, organisation and security in a coherent way.

To the authors' knowledge, ours is the first work bringing RBAC to MAS. Abundant literature exists about role-based approaches for MAS analysis and design [37,9,15], and on the role concept and formalisation in open agent societies [21]. With respect to ours, these approaches focus on the organisation issues, and take in no account – at a model and engineering level – the integration with security, access control and coordination. Also, according to the authors knowledge, this is the first work adopting an algebraic approach to formally describe RBAC architectures. Other formal descriptions have been adopted for RBAC outside the context of MAS, based on set theory [10], Z formal language [16], and temporal logics [7].

In [38], a RBAC-like approach is used to implement a policy-enforcement coordination model based on tuple spaces. In particular, *Proxies* are used for both applying role-based access control on agents interactions – including inter-agent communication and agent-tuple space interaction –, and enforcing coordination policies on tuple spaces. While the ACC abstraction shares some features with the Proxy entity (it is the run-time abstraction responsible of enforcing role-based access control on agent actions), unlike a Proxy, an ACC is negotiated, created and released dynamically to a specific agent, not shared among agents. Also, the ACC is primarily an organisation abstraction: it is used to rule agent actions (protocols) according to its role(s), and not to enforce global coordination policy as in the case of Proxy. Even more, the ACC is meant to specify and enforce not only single actions, but policies, as *patterns* of actions (protocols).

The ACC framework shares some visions and objectives with *policy-driven architectures* [8], developed mostly in the context of distributed systems. These approaches typically make it possible to explicitly define organisation policies – typically role-based – ruling objects interaction and process access to resources inside a distributed system. In this context, the notion of ACC is similar to the notion of *controller* as found in the Law Governed Interaction (LGI) model [19]. Generally speaking, LGI is a message-exchange mechanism that allows an open group of distributed agents to engage in a mode of interaction governed by an explicitly specified and strictly enforced policy – the interaction law of the group. Law enforcement is decentralised, and carried out by a distributed set of controllers, one for each member of the community. As the LGI controller, the ACC enforces rules constraining the action/perception space of the agent exploiting it, enabling the enactment of policy that are local to the agent. With respect to our approach, LGI on the one side is more general, since it is not linked to any specific organisational model – role-based models can be simulated by suitably configuring controllers –; on the other side, in LGI it is not clear how to support dynamism concerning agent organisations, for instance dynamic activation and deactivation of roles.

In the MASs literature, electronic institutions (e-Institutions) currently represent the most complete work (from an engineering point of view) concerning the definition and enforcement of institutional rules and norms governing multi-agent societies and organisations [20,36]. Typically such approaches enforce norms through middle agents, which mediates the communication between the individual agent and the rest of the organisation, without taking into account social rules involving the state of a group of agents/roles. Instead, our approach is rooted on the notion of infrastructure, which we believe can more coherently deal with global system rules (such as role-role relationship) and norms, and with their enforcement as well. In this perspective, the ACC notion can be considered (and exploited) as a run-time embodiment of the notion of *contract* that appeared in some work in the e-Institution and agents & law context [36], or to enforce roles as in [5].

Future work will be devoted to investigate the possibility to integrate research issues about roles, institutions and access-control theory developed in the context of multi-agent organisation within our RBAC-MAS framework – in particular as far as the notions of obligation and normative system are concerned [3]. To this end, the research presented in this paper will soon result in the release of a new version of the TuCSoN infrastructure for MAS coordination [30] fully implementing ACCs and RBAC-MAS.

References

1. Ahn, G., Sandhu, R., Kang, M., Park, J.: Injecting RBAC to secure a web-based workflow system. In: 5th ACM Workshop on Role-based Access Control, pp. 1–10, Berlin, Germany, ACM Press (2000)
2. Bergstra, J. A., Klop, J. W.: Algebra of communicating processes with abstraction. Theor. Comput. Sci. **37**(1), 77–121 (1985)

3. Boella, G., van der Torre, L. W.N. Regulative and constitutive norms in normative multiagent systems. In: 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR'04). pp. 255–266, Whistler, CA, USA (2004)
4. Botha, R. A., Eloff, J. H. P.: Separation of duties for access control enforcement in workflow environments. IBM Syst. J. **40**(3), 666–682 (2001)
5. Cabri, G., Ferrari, L., Leonardi, L.: The RoleX environment for multi-agent cooperation. In: Klusch, M., Ossowski, S., Kashyap, V., Unland, R. (eds.) Cooperative Information Agents VIII, vol. 3191 of LNCS, 8th International Workshop (CIA 2004), Erfurt, Germany, September 27–29, 2004. pp. 257–270. Springer, Berlin Heidelberg New York (2004) Proceedings.
6. Cremonini, M., Omicini, A., Zambonelli, F.: Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In: Garijo, F. J., Boman, M. (eds), Multi-Agent Systems Engineering, vol. 1647 of LNAI, 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain, 30 June – 2 July 1999. pp. 77–88. Springer, Berlin Heidelberg New York (1999) Proceedings.
7. Drouineaud, M., Torrini, M. B. P., Sohr, K.: A first step towards formal verification of security policy properties for rbac. In: 4th International Conference on Quality Software (QSICÆ04). IEEE Press, sep 2004
8. Dulay, N., Damianou, N., Lupu, E., Sloman, M.: A policy language for the management of distributed agents. In: Wooldridge, M., Weiss, G., Ciancarini, P. (eds.) Agent-oriented software engineering II, vol. 2222 of LNCS, pp. 84–100. Springer, Berlin Heidelberg New York (2002)
9. Ferber, J., Gutknecht, O.: A meta-model for analysis and design of organizations in multiagent systems. In: 3rd International Conference on Multi Agent Systems (ICMAS'98). IEEE Press (1998)
10. Ferraiolo, D., Kuhn, R.: Role-Based Access Control. In: 15th NIST–NSA National Computer Security Conference, pp. 554–563, Baltimore, MD, USA, 13–16 October 1992
11. Ferraiolo, D. F., Sandhu, R., Gavrila, S., Richard Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. on Infor. and Syst. Security (TISSEC), **4**(3), 224–274 (2001)
12. FIPA. FIPA communicative act library specification. `http://www.fipa.org`, 2000. Doc. XC00037H.
13. Gelernter, D.: Generative communication in Linda. ACM Trans. Programming Languages and Syst. **7**(1), 80–112 (1985)
14. Kang, M. H., Park, J. S., Froscher, J. N.: Access control mechanisms for inter-organizational workflow. In: 6th ACM symposium on Access Control Models and Technologies, pp. 66–74. ACM Press (2001)
15. Kendall, E. A.: Role modelling for agent systems analysis, design and implementation. IEEE Concurrency, **8**(2), 34–41 (2000)
16. Khayat, A. E., Abdallah, E. J.AND: A formal model for flat role-based access control. In: ACS/IEEE International Conference on Computer Systems and Applications. IEEE Press (2003)
17. Labrou, Y., Finin, T.: Semantics and conversations for an agent communication language. In: Huhns, M. N. Singh, Munindar P. (ed.) Readings in Agents, pp. 235–242. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997)
18. Milner, R.: Elements of interaction: Turing Award lecture. Commun. ACM, **36**(1), 78–89 (1993)
19. Minsky, N. H., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Trans. Software Eng. and Meth. (TOSEM), **9**(3), 273–305 (2000)
20. Noriega, P., Sierra, C.: Electronic institutions: Future trends and challenges. In: Klusch, M., Ossowski, S., Shehory, O. (eds.) Cooperative Information Agents VI, vol. 2446 of LNCS. Springer, Berlin Heidelberg New York (2002)
21. Odell, J., Van Dyke Parunak, H., Brueckner, S., Sauter, J.: Temporal aspects of dynamic role assignment. In: Giorgini, P., Müller, J. P., Odell, J. (eds.) Agent-Oriented Software Engineering IV, volume **2935** of LNCS, 4th International Workshop (AOSE 2003) Melbourne, Australia, July 15, 2003. Revised Papers. pp. 201–213. Springer, Berlin Heidelberg New York (2003)
22. Omicini, A.: Towards a notion of agent coordination context. In: Marinescu, D., Lee, C. (eds.) Process Coordination and Ubiquitous Computing, pp. 187–200. CRC Press (2002)

23. Omicini, A., Ossowski, S.: Objective versus subjective coordination in the engineering of agent systems. In: Klusch, M., Bergamaschi, S., Edwards, P., Petta, P. (eds.) Intelligent Information Agents: An AgentLink Perspective, vol. 2586 of LNAI: State-of-the-Art Survey, pp. 179–202. Springer, Berlin Heidelberg New York (2003)
24. Omicini, A., Ricci, A.: Reasoning about organisation: Shaping the infrastructure. AI*IA Notizie, XVI **2**, 7–16 (2003)
25. Omicini, A., Ricci, A.: MAS organisation within a coordination infrastructure: Experiments in TuCSoN. In: Omicini, A., Petta, P., Pitt, J. (eds.) Engineering Societies in the Agents World IV, vol. 3071 of LNAI, 4th International Workshop (ESAW 2003), London, UK, 29–31 October 2003. Revised Selected and Invited Papers. pp. 200–217. Springer, Berlin Heidelberg New York (2004)
26. Omicini, A., Ricci, A., Viroli, M.: Formal specification and enactment of security policies through Agent Coordination Contexts. Electronic Notes in Theor. Comput. Sci. **85**(3), August 2003. 1st International Workshop "Security Issues in Coordination Models, Languages and Systems" (SecCo 2003), Eindhoven, The Netherlands, 28–29 June 2003. Proceedings.
27. Omicini, A., Ricci, A., Viroli, M.: RBAC for organisation and security in an agent coordination infrastructure. In: Focardi, R., Zavattaro, G. (eds.) 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), pp. 43–62, CONCUR 2004, London, UK, 30 August 2004. Proceedings.
28. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In: Jennings, N. R., Sierra, C., Sonenberg, L., Tambe, M. (eds.) 3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), vol. 1, pp. 286–293, New York, USA, 19–23 July 2004. ACM.
29. Omicini, A., Ricci, A., Viroli, M., Cioffi, M., Rimassa, G.: Multi-agent infrastructures for objective and subjective coordination. Appl Artif. Intell. **18**(9/10), 815–831, (2004). Special Issue: Best papers from EUMAS 2003: The 1st European Workshop on Multi-agent Systems.
30. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems, **2**(3), 251–269 (1999). Special Issue: Coordination Mechanisms for Web Agents.
31. Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.): Coordination of Internet Agents: Models, Technologies, and Applications. Springer, Berlin Heidelberg New York (2001)
32. Ricci, A., Omicini, A., Denti, E.: Activity Theory as a framework for MAS coordination. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) Engineering Societies in the Agents World III, vol. 2577 of LNCS, 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 September 2002. Revised Papers. pp. 96–110. Springer, Berlin Heidelberg New York (2003)
33. Sandhu, R., Coyne, E. J., Feinstein, H. L., Youman, C. E.: Role-based control models. IEEE Comput. **29**(2), 38–47 (1996)
34. Smith, R. G.: The Contract Net Protocol: High-level communication and control in a distributed problem solver. IEEE Trans. Comput. **29**, 1104–1113 (1980)
35. Wegner, P.: Why interaction is more powerful than algorithms. Commun. ACM, **40**(5), 80–91 (1997)
36. Weigand, H., Dignum, V., Meyer, J.-J., Dignum, F.: Specification by refinement and agreement: designing agent interaction using landmarks and contracts. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) Engineering Societies in the Agents World III, vol. 2577 of LNCS, pp. 1–11. Springer, Berlin Heidelberg New York (2003)
37. Zambonelli, F., Jennings, N. R., Wooldridge, M. J.: Organisational rules as an abstraction for the analysis and design of multi-agent systems. Int J Software Eng Knowl. Eng, **11**(3), 303–328 (2001)
38. Zhang, Y., You, J.: An RBAC based policy enforcement coordination model in Internet environment. In: Engineering and Deployment of Cooperative Information Systems, vol. 2480 of LNCS, 1st International Conference (EDCIS 2002), Beijing, China, September 17–20, 2002. Proceedings, pp. 466–477. Springer, Berlin Heidelberg New York (2002)