Role-Based Access Control in MAS using Agent Coordination Contexts*

Alessandro Ricci and **Mirko Viroli** and **Andrea Omicini**
DEIS, Università degli Studi di Bologna
Via Venezia 52
47023, Cesena, Italy
{aricci,mviroli,aomicini}@deis.unibo.it

Abstract

Role-Based Access Control models (RBACs) – and their extensions – are currently considered the most effective approach for engineering access control in complex information systems and dynamic organisations. In this paper we consider their application in the context of Multi-Agent Systems (MAS) – in particular for models and infrastructures supporting role-based organisation models – by means of the notion of Agent Coordination Context (ACC). Here the ACC is used as organisation abstraction released by the MAS infrastructure, defining the runtime context of an agent in terms of its actions/perceptions, according to the roles it is playing inside an organisation. In the paper, we discuss how the ACC abstraction has been used in TuCSoN coordination infrastructure to implement an RBAC-like model, making it possible to dynamically specify and enact articulated role policies, including also interaction protocols.

RBAC-like models for MAS

In contexts such as information systems, Role-Base Access Control (RBACs) models are currently considered the most effective way to engineer security for complex organisations (Ferraiolo & Kuhn 1992; Sandhu *et al.* 1996). Their major properties concern the ability to articulate and enforce enterprise (system) specific security policies and to streamline the burdersome process of security management (Ferraiolo & Kuhn 1992). With respect to the previous approaches (discretionary and mandatory access control), they allow for more flexible and detailed control and management of security.

Because of this flexibility, extensions have been layered on top of the basic RBAC model for effectively integrating security and organisation issues in the context of open, dynamic and complex systems, such as inter-organisational workflow management (Kang, Park, & Froscher 2001) and pervasive computing environments (Tripathi *et al.* 2004). These are application scenarios that – given their complexity – are suitable to be designed and developed using the agent paradigm, in particular for what concern the coordination and organisation issue (Ricci, Omicini, & Denti 2002). We claim then that RBAC-like model can be suitably introduced in MAS models & infrastructures, integrating a high level role-based security (access control) approach with MAS coordination and organisation, where the role abstraction is already at play.

MAS organisational models based on roles are typically exploited as analysis and design tools, in particular in AOSE methodology: conversely, conceiving an RBAC-like model into MAS shifts (or completes) the focus on the runtime aspects: roles, sessions, policies become runtime issues of a MAS organisation, manageable dynamically by suitable services provided by the infrastructure. More generally, RBAC approaches have brought at the infrastructure level security issues which previously were faced (in a similar way) by each individual applications; analogously, we aim at factorising security issues that frequently emerge in the engineering of a distributed systems in terms of agents, extending the MAS infrastructure with suitable services, integrated with the MAS organisation/coordination model.

So, the question is: how to introduce an RBAC-like approach in MAS (models/infrastructures)? And, how can it be integrated with possible MAS role-based models?

In this paper we provide an answer based on the notion of Agent Coordination Context (ACC). Briefly, ACCs have been introduced as a mean to model explicitly presence of an agent inside an (organisational) environment, both in terms of the actions/perceptions the agent can do (as a kind of environment interface) and their effects. In this paper, the ACC abstraction – and related infrastructure support – is shown to be effective for embeding an RBAC-like model in the context of TuCSoN, a MAS coordination infrastructure (Omicini & Zambonelli 1999).

The remainder of the paper is organised as follows: in the first section we provide an overview of the RBAC approach and of the reference architecture considered a standard in literature; in the second section, after recalling the notion of ACC, we discuss its application to develop an RBAC-like model in MAS, in particular describing how the basic RBAC reference architecture is mapped using an ACC-based approach. Then, in the third section we discuss a concrete

*Research under the auspices of: MIUR (the Italian Ministry of Education, University and Research), COFIN 2003 project "Fiducia e diritto nella società dell'informazione", paper no. 4; and EC (the European Community), FP6 project "AgentLink III".

realisation of the ACC framework in TuCSoN infrastructure, so as to have an RBAC-like model in the context of our coordination infrastructure. Finally, in the last section we provide the conclusions, briefly discussing also related and future work.

RBAC Models Overview

In RBAC, a *role* is properly viewed as a semantic construct around which access control policy is formulated, bringing together a particular collection of users and permissions, in a transitory way (Sandhu *et al.* 1996). The role concept assumes several manifestations, which RBAC aims at accomodate and capture. A role can represent competency to do specific tasks, such as a physician or a pharmacist; but also the embodiement of authority and responsibility, such as in the case of a project supervisor. Authority and responsibility are distinct from competency.

The RBAC approach provides the capability to establish relations between roles as well as between permissions and roles and between users and roles. For example, two roles can be established as mutually exclusive, so the same user is not allowed to take on both roles. 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 must be invoked to complete a sensitive task, such as requiring an accounting clerk and account 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 by the operating system. In spite of the support for these principles, RBAC is said to be *policy neutral*, since it does not enforce itself any specific access policy.

Summing up, RBAC provides an encapsulation of security policy. Access control strategy is encapsulated in various components of RBAC such as role-permission, user-role and role-role relationships. These components, configurable dynamically by system administrators, collectively determine whether a particular user will be allowed to access a particular piece of data in the system. Moreover, RBAC approach makes it possible & easy to incrementally evolve the access control policy over the system life cycle, to meet the changing needs of an organisation.

The Reference Architecture

According to the reference architecture formally defined in (Ferraiolo *et al.* 2001), the main components of an RBAC model are depicted in Figure 1, 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 (`SESSIONS`).

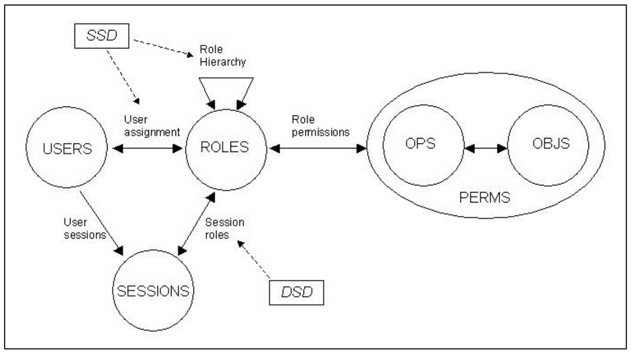

Figure 1: RBAC Reference Model (Ferraiolo *et al.* 2001)

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 (Ferraiolo *et al.* 2001). 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 to reflect an organisation's line of authority and responsibilities, 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 of 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.

Agent Coordination Contexts for RBAC

In this section we discuss how we exploited the Agent Coordination Context (ACC) abstraction to port an RBAC-like model in the context of MAS. Before going into details of the RBAC-like architecture based on ACC, it is useful to recall the basic features of the ACC abstraction.

The Notion of Agent Coordination Context

The notion of ACC has been introduced in (Omicini 2002) as a way to model MAS environment from the individual agent viewpoint, more precisely the *objective* and *subjective* relationships between an agent and the (organisational) environment in which he is immersed.

As reminded by the name, an ACC is meant to represent a *context*, factorising the many meanings that emerge from the several research areas where it is commonly used (language, philosophy, logic, artificial intelligence, ...): an abstraction aimed at modelling the effect of the environment – in its most general acceptation, including the spatial and temporal interpretation of the terms – on the interaction and communication occurring among active (and typically intelligent) entities, such as humans or artificial agents. Within agent societies, the notion of context can be used as a first-class abstraction for modelling and engineering the environment. The full characterisation of ACC is obtained then by identifying the context abstraction as the conceptual place where to set the boundary between the agent and the environment, so as to encapsulate the *interface* that enables agent actions and perceptions inside the environment.

More precisely, an ACC (i) works as a model for the agent environment, by describing the environment where an agent can interact, and (ii) enables and rules the interactions between the agent and the environment, by defining the space of the admissible agent interactions.

A useful metaphor can be adopted as a conceptual example of an ACC, the *control room metaphor* (Omicini 2002). According to this metaphor, an agent entering a new environment is assigned its own control room, which is the only way in which it can perceive the environment, as well as the only way in which it can interact. The control room offers the agent a set of admissible inputs (lights, screens,...), admissible outputs (button, cameras,...). How many input and output devices are available to an agents, of what sort, and for how much time is what defines the control room *configuration*, that is the specific ACC. More insights on the ACC concept can be found in (Omicini 2002).

ACC for MAS Organisation

So, this characterisation makes the ACC a suitable abstraction modelling the *presence* or position of an agent within an organisation, in terms of its admissible actions with respect to organisation resources and its admissible communications toward the other agents belonging to the organisation.

Two basic stages characterise the ACC dynamics: *ACC negotiation* and *ACC use*. An ACC is meant to be negotiated by the agents with the MAS infrastructure, in order to start a *working session* inside an organisation. The agent requests an ACC specifying which roles to activate. If the agent request is compatible with (current) organisation rules, a new ACC is created, configured according to the characteristics of the specified roles, and then released to the agent for active playing inside the organisation. The agent then can use the ACC to interact with the organisation environment, by exploiting the actions/perceptions enabled by the ACC.

ACC for RBAC

In the context of the RBAC reference architecture, the ACC naturally embodies the concept of session, coupling dynamically agents (as users) and the organisation environment. Then, ACCs represent the runtime entities that physically enable and constraint agent actions, according to rules which in the RBAC framework correspond to the role-permission relationships.

Figure 2 shows the use of the ACC for realising an RBAC architecture. As already mentioned, agents are the users of the systems and the ACCs play the role of sessions. An agent

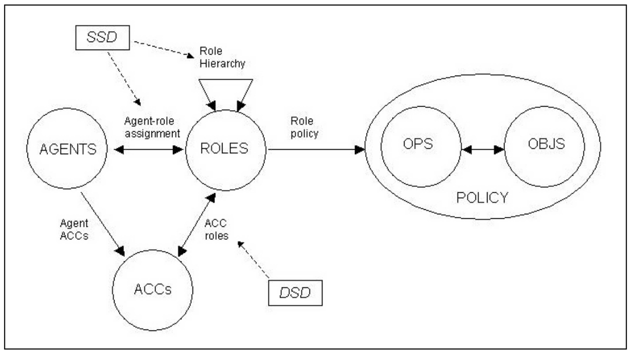

Figure 2: An RBAC-like Model using ACCs

can hold multiple ACCs at a time, for each organisation in which he is playing; the same ACC can involve the activation of multiple roles, and the same role can be played simultaneously in different ACC.

ACC negotiation is ruled by both currently defined agent-role relationships (defining the SSD rules in Figure 2) and inter-role relationships (used for defining DSD rules). The former are described with rules defining static agent-role assignment, so as to specify which agents are allowed to play a role or what credentials or characteristics they must exhibit. The latter, instead, describe constraints on agent entrance, considering the dynamic roles which the agent aims to activate; examples of relatioships typically used for this purpose are role exclusion rules (if an agent has activated a role *R1*, he can't activate a role *R2*) and role inclusion rules (in order to activate the role *R1* the agent must have activated also the role *R2*).

Once the properly configured ACC has been released to the agent (after a successful negotiation), the relationships defined among roles and policies as depicted in Figure 2 shape the agent action space enabled by the ACC, defining what actions – as operations within the environment – the agent is allowed to execute. Actually, the ACC model makes it possible to specify not only rules on the individual actions, but also articulated patterns of actions and interaction protocols, introducing also timing constraints. For this purpose, a formal semantics of ACC based on process algebra has been defined (Omicini, Ricci, & Viroli 2003), providing a formal specification of the individual role policies and of their composition.

Experiments in TuCSoN

The ACC abstraction has been exploited to bring an RBAC-like model in the TuCSoN coordination infrastructure: In TuCSoN the organisation resources accessed by agents are *tuple centres*, coordination media (services) provided by the infrastructure to support coordination activities of agents (Omicini & Zambonelli 1999). Tuple centres are thought to mediate agent access to organisation resources (e.g. printer), acting as a kind virtual proxies of the resources and embedding coordination policies governing their exploitation (Omicini & Ossowski 2003).

Technically, tuple centres are *programmable tuple spaces*

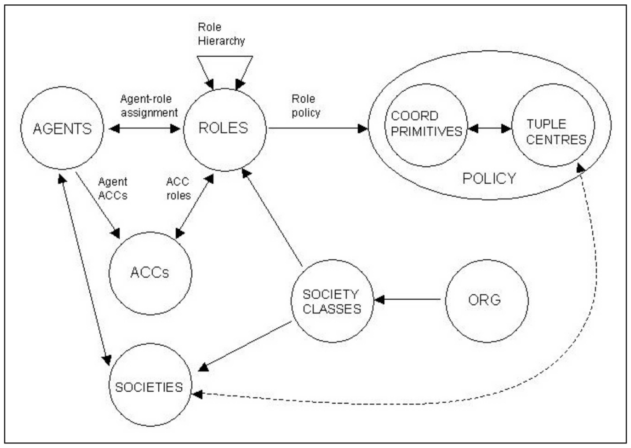

Figure 3: An RBAC-like Model using ACCs in TuCSoN

– sort of reactive, logic-based blackboards that agents access associatively by writing, reading, and consuming *tuples* – ordered collections of heterogeneous information chunks – via simple communication operations (*out*, *rd*, *in*, *inp*, *rdp*). While the behaviour of a tuple space in response to communication events is fixed, the behaviour of a tuple centre can be tailored to the application needs by defining a set of *specification tuples* expressed in the ReSpecT language, which define how a tuple centre should react to incoming/outgoing communication events (Omicini & Denti 2001). So, differently from tuple spaces – and related implementation such as JavaSpaces (Freeman, Hupfer, & Arnold 1999) or T-Spaces (Wyckoff *et al.* 1998)– tuple centres can be programmed with reactions so as to encapsulate coordination laws directly in the coordination media. In other words, tuple centres can be conceived as general-purpose customisable *coordination artifacts*, whose behaviour can be dynamically specified, forged and adapted so as to support the coordination activities among agents (Ricci, Omicini, & Denti 2003; Omicini *et al.* 2004). From a topological point of view, tuple centres are collected in the node of the infrastructures, spread over the network.

Figure 3 shows the RBAC-like architecture specialised in the TuCSoN case. Tuple centres are the objects, and the tuple centre coordination primitives are the operations. The architecture includes also abstractions introduced in TuCSoN to model and structure organisations (Omicini & Ricci 2003): a system is defined as an organisation (ORG in Figure 3), dynamically structured in societies, as instances of society classes (acting as a template). A society class groups a set of roles; each role is characterised by a policy, defining the actions and interaction protocols allowed for agents playing such a role. An agent action has the form: `Tid @ Node ? op`, where `Tid` is the tuple centre identifier, `Node` is the TuCSoN node hosting the tuple centre and `op` is the coordination primitive. So, permissions as found in the RBAC model are expressed here in terms of rules on allowed actions/interactions and their aggregation as protocols – i.e. role policies.

Societies are dynamically composed by a set of agents and tuple centres. An agent takes part actively to an organisation by playing at least one role in a society.

From the dynamic point of view, an agent negotiates with the infrastructure service of an organisation an ACC specifying which roles to activate in which societies. The agent request is accepted only if it is compatible with the agent-role assignment and the inter-role relationships. After a successful negotiation, the agent receives an ACC whose policy reflects the composition of the policies defined for the individual roles activated.

It is worth noting that ACCs are meant to be dynamically negotiated with an organisation, as well as – dynamically – it is possible for an agent to activate/deactivate roles, according to the constraints and rules defined by specific organisations. Also, the set of available roles is open: roles can be added/removed/changed dynamically by actors (humans as well as agents), playing administration roles with proper permissions (enabled by suitable ACCs).

Role Policy Description

In TuCSoN the role policies are described using a Prolog theory, as a set of rules defining what patterns of actions and interactions are allowed. The theory – which is meant to be (dynamically) specified by the agent/human administrator of an organisation – is composed by a set of can_do rules defining role action space:

```
can_do(CurrentState,Action,NextState):- Conditions.
```

This rule means that the action `Action` can be executed in the role state `CurrentState` if conditions `Conditions` value, and – in that case – next role state is `NextState`. Note that each can do rule can be seen as the rule of a labelled transition systems (Omicini, Ricci, & Viroli 2003).

The concept of role state is used as a way to easily express interaction protocols; any Prolog term – also structured, partially specified – can be used to denote the role state. By default, the starting state is denoted by the `init` atom. `CurrentState` and `NextState` can be omitted, using the *any* Prolog symbol (_): omitting the `CurrentState` information accounts for stating the validity of the rules for every possible state; omitting the `NextState` information accounts for keeping current state as next state. Finally, `Action` denotes an agent action, as described in previous subsection.

Actually, `Conditions` can contain also built-in predicates useful to describe context-aware (with respect to local time, space and identity/positions of the agents) policies. Among the predicates, here we mention: `agent_id(-ID)` (which retrieves the identity of the agent owner the ACC), `local_node(-Node)` (retrieves the node hosting the agent), `session_time(-TimeInMillis)` (which retrieves the number of milliseconds passed since the release of the ACC)[1].

So, given a role policy theory, an action is admitted for the specific role if and only if there is a `can_do` rule which holds for it; in other words, an action `Action` is allowed if the goal `can_do(+CurrentState,+Action,-NextState)` can be

[1]This value refers to the time in which the ACC accepts the request execution for the action.

demonstrated, given the theory. So, for instance a rule of the kind: `can_do(_,_,_).` means that every action is allowed. `can_do(_,Action,_).` means that the action *Action* can be always executed.

By specifying the body of the rules it is possible to express also specific forbidden actions: `can_do(_,Act,_):- not(Act=Action).` means that every action can be executed but actions matching with *Action* template. Actually, this Prolog encoding makes it easy to map the formal language based on process algebras defined in (Omicini, Ricci, & Viroli 2003).

The ACC policy is defined by composing the individual policies of the roles which the ACC represents. Actually, the Prolog theory defining the ACC overall policy is obtained by simply composing the individual theories, and adding the meta-rules which define composition:

```
can_do(comp(S1,S2),Action,comp(S1Next,S2)):-
    can_do(S1,Action,S1Next).
can_do(comp(S1,S2),Action,comp(S1,S2Next)):-
    can_do(S2,Action,S2Next).
```

`comp(S1,S2)` is the logic state of an ACC composing two roles. The composition property is used recursively: for three roles we have `comp(S1,comp(S2,S3))`, for four `comp(S1,comp(S2,comp(S3,S4)))`, and so on. The semantics of the composition is: an action is allowed if and only if it is allowed according to the policy of the first role or, if this condition is not satisfied, of the second. This is the semantics of the parallel operator as defined in process algebras, with a difference: the approach adopted here implicitly specifies an order between the roles, affecting also the order in which the rules are considered (which is not the case in process algebras).

Simple Examples

Suppose to setup a blackboard society, where any agent can insert and read tuples `msg(M)` as messages on the `bboard` tuple centre, but only the administrators can remove them. So, a suitable role policy for users can be :

```
    can_do(_, bboard ?  out(msg(_)),_).
    can_do(_, bboard ?  rd(msg(_)),_).
```

Instead for administrators:

```
    can_do(_, bboard ?  _,_).
```

Then, consider a slightly variation of previous example, in which the tuple centre `msg_box` is used to exchange messages `msg(DestID,Content)`. Any user can send messages to any other one, but can retrieve only messages sent to him. A suitable user policy can be then:

```
    can_do(_, msg_box ?  out(msg(DestID,Content)),_).
    can_do(_, msg_box ?  in(msg(DestID,Content)),_):-
                  agent_id(DestID).
```

An ACC with a lease time *T* can be easily modelled with the rule:

```
    can_do(_, _, _):- session_time(ST), ST < T.
```

Finally, as example of context-aware policy:

```
    can_do(_, Tid ?  out(_), _):- local_node(Node).
```

The rule asserts that only agents located at the node *Node* can insert tuples into the tuple centre *Tid*.

A Case Study: Contract Net Protocol

For sake of concreteness, we briefly illustrate the role policies with a simple version of a well-known agent interaction protocol, the Contract Net (CNP) (Smith 1979).

Suppose to be in a task allocation scenario, inside an organisation called `acme.org`. A society for task allocation coordination activity is defined, called `task_distribution`. The society involves two roles, `master` and `worker`, and a coordination artifact used to support the task allocation coordination activity, the `tasks` tuple centre. This tuple centre encapsulates the coordination rules involved in the task allocation, actually embodying a CNP-like protocol, where masters are the managers and workers are the contractors. The coordinating behaviour of the tuple centre is defined by the reactions shown in Table 1 (bottom) expressed in the ReSpecT language, and is breafly described in the caption of the table.

The interactive behaviour of masters and workers is described in the pseudo-code in Table 1 (top). Agents aiming at playing the role of masters negotiate an ACC with the `master` role, instead workers specify the `worker` role (line 0). A master announces a task to be performed (line 1) and waits for bids: when the related timeout expires, it retrieves the list of bids from potential contractors (line 3), selects the best bid according to its evaluation (line 4), awards the contract (line 5), and receives the result of the performed task (line 6).

On the other side, a worker (Table 1, right) waiting for possibly-interesting task announcements (line 1) evaluates its capability to respond to this request (line 2), issues its bid, and starts waiting for an answer. If the bid is accepted by the master (answer `awarded`), the worker performs the task (line 5) and outputs the computed result (line 6).

Table 2 (top) shows a possible role policy for the master role, specified by the society administrator and enacted by the ACC. The policy actually enforces an interaction protocol, composed by four different role states, which correspond to different protocol stages. Among the constraints, the master agent can access only information concerning the task he announced, both when reading / getting the bid list, when announcing the winner and when retrieving the results. The same artifact could then support several task allocation activities, with proper separation of duty among masters. Also, the policy constrains master agents to award an agent which was necessarily among the bidders of the specific task announcement.

Conversely, Table 2 (bottom) shows a possible role policy for the bidder. Also for the bidder the policy enforces a protocol composed basically by two states (and related proto-

```
enterACC('acme.org', [role(master,task_distribution)])
tasks ?  out(announcement(Task))
wait(ExpireTime)
tasks ?  in(bids(Task,BidList))
Bid ← selectWinner(BidList)
tasks ?  out(award(Task,Bid))
tasks ?  in(result(Task,Result)
```

```
enterACC('acme.org',[role(worker,task_distribution])
tasks ?  rd(announcement(Task))
MyBid ← evaluate(Task)
tasks ?  in(bid(Task,MyBid,Answer))
if (Answer=='awarded') {
    Result ← perform(Task)
    tasks ?  out(result(Task,Result)}
```

```
1 reaction(in(bid(Task,MyBid,Answer)), (
    pre,out_r(contractor(Task, MyBid)),
    in_r(bids(Task, L)),
    out_r(bids(Task, [MyBid|L])) )).

2 reaction( out(announcement(Task) ), (
    out_r(bids(Task,[])) )).

3 reaction( in(bids(Task,L) ), ( post,
    in_r(announcement(Task)) )).

4 reaction( out(award(Task,TheBid) ), (
    in_r(award(Task,TheBid)),
    in_r(contractor(Task,TheBid)),
    out_r(bid(Task,TheBid,awarded)),
    out_r(refuse_others(Task)) )).
```

```
5 reaction(out_r(refuse_others(Task)), (
    in_r(refuse_others(Task)),
    in_r(contractor(Task,TheBid)),
    out_r(bid(Task,TheBid,'not-awarded')),
    out_r(refuse_others(Task)) )).

6 reaction(out_r(refuse_others(Task)), (
     in_r(refuse_others(Task)),
     no_r(contractor(_,_)) )).
```

Table 1: *(Top)* The behaviour of master (left) and worker (right) agents in the CNP example *(Bottom)* The ReSpecT code defining the behaviour of the tuple centre `tasks`, implementing the coordination rules of the Contract Net Protocol. A brief explanation: When a master issues a task announcement, reaction 2 is triggered and coordination data are set up in the g. Each time a worker makes a bid, reaction 1 stores information about the new proposal and updates the bid list. When the master eventually collects the bids, reaction 3 removes the task announcement tuple: so, no more bids are considered. Finally, when a master awards the contract, reaction 4 emits the tuple `bid(Task, TheBid, awarded)` to notify the specific worker waiting for that answer, then triggers reaction 5 and 6: the first places the `refuse_others` tuple, which is used to collect and remove the information about other bidders (tuples `contractor`), while reaction 6 notifies the other contractors by emitting the `bid(Task, TheBid, 'not-awarded')` tuple.

```
can_do(init, tasks ?  out(announcement(Task)), task_announced(Task)).
can_do(task_announced(Task), tasks ?  rd(announcement(Task)),_).
can_do(task_announced(Task), tasks ?  rd(bids(Task),_)),_).
can_do(task_announced(Task), tasks ?  in(bids(Task),BidList)),choose_bidder(Task,BidList)).
can_do(choose_bidder(Task,BidList), tasks ?  out(award(Task,Bid)), get_result(Task,Bid)):- element(Bid,BidList).
can_do(get_result(Task,_)), tasks ?  rd(result(Task,_)),_).
can_do(get_result(Task,_)), tasks ?  in(result(Task,_)),init).
```

```
can_do(init, tasks ?  rd(announcement(_)),_).
can_do(init, tasks ?  in(task(Task,_,awarded)),awarded(Task)).
can_do(awarded(Task), tasks ?  out(result(Task,_),init)).
```

Table 2: Role policy for the master role *(Top)* and worker role *(Bottom)*.

col stages). Moreover, the constraints provided by the ACC avoid non-awarded bidders (workers) to provide fake results of the task to be allocated: only the awarded bidder is allowed to insert the tuple concerning task results.

Of course the example considers only a simplified version of the contract net: more complex cases – for instance involving time constraints – would need more articulated role policy specifications.

Conclusion

In this paper we introduced the RBAC model for engineering some security and organisation aspects in the context of MAS. For this purpose, we exploited the Agent Coordination Context notion to extend the TuCSoN coordination infrastructure with an RBAC-like architecture.

Adopting an RBAC-like approach makes it possible to gain all the benefits of the approach in engineering security inside complex MAS organisations, mainly in terms of encapsulation of the security policies, and flexibility in their management. ACCs have been the key for porting the model on top of our existing infrastructure, integrating coordination, organisation and security in a coherent way. The resulting security & organisation model provides features that are essential for the engineering of open MAS, namely:

- *Dynamism/Flexibility*. Agents can enter and exit dynamically from organisations, activativating/deactivating roles by (re)negotiating ACCs. Also, the organisation structure is meant to be changeable/adaptable at runtime, by adding and removing roles (societies and society classes) and changing the role policy.
- *Support for Heterogeneity*. Analogously to TuCSoN coordination model, the RBAC-like model is neutral with respect to the specific agent computational model and platform. Then, the approach can be used to define and enforce access control both for reactive and intelligent/cognitive agents, belonging to different agent platform and using TuCSoN for coordination purposes.
- *Formal Property Verification*. The RBAC architecture makes it easy to conceive a framework for verification of security & organisation properties, by well separating and encapsulating security policies. Properties concerning role activation and separation of duties can be verified focussing on the agent-role and role-role relationships, ruling the ACC negotiation process. In our framework this will be possible after a formalisation of the organisation model (structures and rules) reported in (Omicini & Ricci 2003), which is an ongoing work. Instead, properties concerning access control and, more generally, the safe/correct execution of interaction protocols can be verified focusing on ACC behaviour, as composition of individual role policies. This is already possible, given the formal semantics defined for ACC (Omicini, Ricci, & Viroli 2003).

According to the knowledge of the authors, this is the first work considering the application of RBAC models in the context of MAS. Abundant literature exists concerning role-base approaches for MAS analysis and design (Zambonelli, Jennings, & Wooldridge 2001; Ferber & Gutknecht 1998; Kendall 2000), on the role concept and formalisation in open agent societies (Odell, Parunak, & Fleischer 2003; Dastani, Dignum, & Dignum 2003; Odell *et al.* 2003), and for MAS development and runtime (Ferber, Gutknecht, & Michel 2003; Cabri 2001). Mainly, these approaches are focussed on the organisation issues, without taking into the account – at a model and engineering level – the integration with security and access control.

Roles and organisation/social rules are main issues also in the context electronic Institutions (Esteva *et al.* 2000): actually, we aim at investigating the application of our approach also in that context, providing an infrastructure with first class entities (coordination artifacts and ACCs) for modelling, specifying and enacting the role of the institution.

Agent Coordination Contexts are somewhat similar in their ruling and controlling action to *Controllers* as defined in Law Governed Interaction (LGI) (Minsky & Ungureanu 2000). However, the LGI does not consider explicitly any role based model, and concerns mostly (low level) distributed systems, rather than MASs.

Future work accounts for moving from theory to practice, completing the implementation of the TuCSoN extension with ACCs and the RBAC-like model, and stressing the validity of the model by implementing MASs on top of it. In particular, we will reconsider systems previously engineered on top of TuCSoN, such as distributed workflow management systems (Ricci, Omicini, & Denti 2002), and the we will re-engineer them – or better, their organisation and security asset – with the new RBAC support. Also, future work will be devoted to continue the formal investigation of these issues, adding to the formalisation of the ACC the formal specification of the RBAC architecture as defined in this paper.

References

Cabri, G. 2001. Role-based infrastructures for agents. In *Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2001)*. IEEE.

Dastani, M.; Dignum, V.; and Dignum, F. 2003. Role-assignment in open agent societies. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 489–496. ACM Press.

Esteva, M.; Rodrguez-Aguilar, J. A.; Arcos, J. L.; Sierra, C.; and Garcia, P. 2000. Institutionalising open multi-agent systems. In *Proceedings of the 4th International Conference on MultiAgent Systems (ICMAS 2000)*, 381–383.

Ferber, J., and Gutknecht, O. 1998. A meta-model for analysis and design of organizations in multi-agent systems. In *Proceedings of ICMAS '98*. IEEE Press.

Ferber, J.; Gutknecht, O.; and Michel, F. 2003. From agents to organisations: an organizational view of multi-agent systems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*. Melbourne, Australia: ACM Press.

Ferraiolo, D., and Kuhn, R. 1992. Role-based access control. In *Proceedings of the NIST–NSA National (USA) Computer Security Conference*, 554–563.

Ferraiolo, D. F.; Sandhu, R.; Gavrila, S.; Kuhn, D. R.; and Chandramouli, R. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 4(3):224–274.

Freeman, E.; Hupfer, S.; and Arnold, K. 1999. *JavaSpaces: Principles, Patterns, and Practice*. The Jini Technology Series. Addison-Wesley.

Kang, M. H.; Park, J. S.; and Froscher, J. N. 2001. Access control mechanisms for inter-organizational workflow. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, 66–74. ACM Press.

Kendall, E. A. 2000. Role modelling for agent systems analysis, design and implementation. *IEEE Concurrency* 8(2):34–41.

Minsky, N. H., and Ungureanu, V. 2000. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 9(3):273–305.

Odell, J.; Parunak, H. V. D.; Brueckner, S.; and Sauter, J. 2003. Temporal aspects of dynamic role assignment. In Giorgini, P.; Müller, J. P.; and Odell, J., eds., *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, 201–213. Springer.

Odell, J.; Parunak, H. V. D.; and Fleischer, M. 2003. The role of roles in designing effective agent organizations. In Garcia, A.; Lucena, C.; Zambonelli, F.; Omicini, A.; and Castro, J., eds., *Software Engineering for Large-Scale Multi-Agent Systems*, volume 2603 of *Lecture Notes in Computer Science*, 27–28. Springer.

Omicini, A., and Denti, E. 2001. From tuple spaces to tuple centres. *Science of Computer Programming* 41(3):277–294.

Omicini, A., and Ossowski, S. 2003. Objective versus subjective coordination in the engineering of agent systems. In Klusch, M.; Bergamaschi, S.; Edwards, P.; and Petta, P., eds., *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*. Springer-Verlag. 179–202.

Omicini, A., and Ricci, A. 2003. Reasoning about organisation: Shaping the infrastructure. *AI*IA Notizie* XVI(2):7–16.

Omicini, A., and Zambonelli, F. 1999. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems* 2(3):251–269.

Omicini, A.; Ricci, A.; Viroli, M.; and Castelfranchi, C. 2004. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*. New York, USA: ACM Press.

Omicini, A.; Ricci, A.; and Viroli, M. 2003. Formal specification and enactment of security policies through Agent Coordination Contexts. In Focardi, R., and Zavattaro, G., eds., *Security Issues in Coordination Models, Languages and Systems*, volume 85(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V.

Omicini, A. 2002. Towards a notion of agent coordination context. In Marinescu, D., and Lee, C., eds., *Process Coordination and Ubiquitous Computing*. CRC Press. 187–200.

Ricci, A.; Omicini, A.; and Denti, E. 2002. Virtual enterprises and workflow management as agent coordination issues. *International Journal of Cooperative Information Systems* 11(3/4):355–379.

Ricci, A.; Omicini, A.; and Denti, E. 2003. Activity Theory as a framework for MAS coordination. In Petta, P.; Tolksdorf, R.; and Zambonelli, F., eds., *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*. Springer-Verlag. 96–110. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 September 2002. Revised Papers.

Sandhu, R.; Coyne, E. J.; Feinstein, H. L.; and Youman, C. E. 1996. Role-based control models. *IEEE Computer* 29(2):38–47.

Smith, R. G. 1979. The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, 186–192. Washington D.C.: IEEE Computer Society.

Tripathi, A.; Ahmed, T.; Kulkarni, D.; Kumar, R.; ; and Kashiramka, K. 2004. Context-based secure resource access in pervasive computing environments. In *Proceedings of the 1st IEEE International Workshop on Pervasive Computing and Communications Security(IEEE PerSec'04)*. IEEE.

Wyckoff, P.; McLaughry, S. W.; Lehman, T. J.; and Ford, D. A. 1998. T Spaces. *IBM Journal of Research and Development* 37(3 - Java Techonology):454–474.

Zambonelli, F.; Jennings, N. R.; and Wooldridge, M. 2001. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering* 11(3):303–328.