RBAC-MAS & SODA: Experimenting RBAC in AOSE

Ambra Molesini[1], Enrico Denti[1], and Andrea Omicini[2]

[1] Alma Mater Studiorum—Università di Bologna
viale Risorgimento 2, 40136 Bologna, Italy
ambra.molesini@unibo.it, enrico.denti@unibo.it

[2] Alma Mater Studiorum—Università di Bologna a Cesena
via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it

Abstract Role-Based Access Control models are currently considered as the most effective approach for engineering access control systems. In this paper we experiment their application in the context of Multi-Agent Systems (MAS), by discussing the design of an access control system with an agent-oriented methodology such as SODA. In particular, we show how a clear separation between mechanisms and policies can be achieved by organising the access control system along two layered subsystems, and discuss the advantages of such an approach.

1 Introduction

According to Anderson, "security engineering is about building systems to remain dependable in the face of malice, error or mischance" [1]. Security requirements are often still considered among the non-functional system requirements—something that can be taken into account at a later point of the software development process. Yet, fitting security mechanisms into an existing design leads to design problems and software vulnerabilities: security should rather be considered as a key issue throughout the whole development process, to be defined and explored in the requirements specification phase—i.e., among the functional requirements [2]. So, software engineering methodologies should provide developers with models and processes able to effectively capture the security concerns.

In principle, the agent-oriented paradigm seems a good candidate for capturing security issues in software systems, since the intrinsic agents' features – such as autonomy, intentionality and sociality – make it possible to express the security requirements at the proper abstraction level at early stages of the engineering process, mapping them onto suitable security mechanisms in subsequent stages. A software system can be then conceived as a Multi-Agent System (MAS) where autonomous entities – the agents – interact with each other in order to achieve their goals: such an agent "ensemble" is often represented as an agent *society* at design time. Thus, software engineering techniques should specifically be inspired by – and tailored to – the agent paradigm, so as to fully exploit the

agent as well as the agent society metaphors: according to that, several agent-oriented methodologies have been proposed in the last years [3,4,5].

However, most of them still fall short in providing a full-fledged security-oriented approach for agent-oriented systems, although some recent research work is opening the way especially with respect to the requirements analysis [6,7]. Indeed, research on security for MAS has been mainly focused on the solution of individual security problems, such as attacks from an agent to another, from a platform to an agent, and from an agent to a platform [2]: instead, adequately integrating security and systems engineering into a coherent agent-oriented development process is seemingly a more complex task. In particular, in the engineering of interaction of complex software systems, security is strictly related to two other dimensions—coordination and organisation [8,9,10]. In fact, both coordination and security establish laws and rules for constraining the space of interaction—that is, the system dynamics; organisation is their static counterpart, in that it specifies on the one side the agents and the roles they play, as well as the agent-role and inter-role relationships between them. The connection between organisation and security is quite apparent in Role-Based Access Control (RBAC) models & architectures, currently considered as the most promising approach in the engineering of security of complex information systems [11].

Generally speaking, access control is aimed at allowing authorised users to access the system resources they need, while preventing unauthorised users to do the same. Today, access control is typically designed by clearly separating the definition of a suitable *access policy* – i.e., the set norms for granting / refusing access to resources – from the hardware & software *mechanisms* used to implement and enforce it: this separation guarantees independence between the protection requirements to be applied and the way they are applied. Different access policies can thus be easily compared independently from their actual implementation, and changed with no impact over the system; in its turn, the underlying mechanism can support different, multiple policies over time.

More recently, the RBAC technique has been introduced in the context of MAS infrastructures, integrating a role-based security approach with agent-based coordination and organisation, where the role abstraction is already at play [8]. This choice helps facing the typical MAS heterogeneity and openness, since the security properties can be specified in terms of RBAC general concepts: for instance, participating agents can adopt heterogeneous computational models (from purely reactive to cognitive ones), as well as enter/ leave/ change role in the organisation as needed, according to the system policies.

In this paper we first discuss the application of RBAC models in the MAS context (Section 2). Then, we focus on the RBAC requirements to be addressed for engineering an RBAC system (Subsection 2.2), and show how SODA, an agent-oriented methodology, addresses those requirements (Section 3) Finally, as a case study, we apply our approach to the control of the accesses to a university building (Section 4), and show the benefits of a clear separation between mechanisms and policies. Discussion and comparison with some relevant related work are reported in Section 5.

2 Access Control in Multi-Agent Systems

Access control is aimed at enabling (only) the authorised users to access the system resources in a controlled and supervised way. A key aspect is the clear separation between the rules used to decide whether access to a resource should or not be granted for a given user – the *access policy* – and the hardware & software *mechanisms* actually enforcing such rules. Such a separation is useful for two main reasons: first, to uncouple the definition of a policy from its implementation, so that the latter is not affected by policy changes; then, to more easily identify the basic properties that any access control system should satisfy—such as complete mediation, default deny, minimum privilege, etc.

With respect to previous access control models, such as Discretionary Access Control (DAC) and Mandatory Access Control (MAC), RBAC – a NIST standard [12] – specifies security policies in terms of organisational abstractions (users, roles, objects, operations, permissions, and sessions) and their relationships [13]. Users are assigned to roles, and roles to permissions. 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 that plays the role at a given time. A *permission* is an approval to perform an *operation* on some protected *objects*: the exact semantics of "operation" and "object" depends on the specific case. A session is a mapping between a given user and the subset of its currently active roles: so, each session is associated with a single user, while a user can be associated to one or more sessions.

Organisation rules are defined in terms of relationships between the above elements—namely, between roles and permissions, and between roles and users; inter-role relationships are also introduced to specify *separation of duties*. More precisely, *static separation of duty* (SSD) is obtained by enforcing constraints on the assignment of users to roles, while *dynamic separation of duty* (DSD) is achieved by placing constraints on the roles that can be activated within or across the given users' session(s). Accordingly, introducing RBAC in the context of MAS coordination models and infrastructures essentially amounts at mapping roles, sessions, and policies onto suitable runtime issues of the MAS organisation, dynamically manageable via infrastructural services.

2.1 RBAC for Multi-Agent Systems

RBAC-MAS [9,14] is a model for an RBAC-like organisation of MAS, where RBAC general concepts are tailored to MAS specificities (Figure 1).

Generally speaking, agent-oriented methodologies often exploit MAS role-based organisational models just as analysis & design tools [15]: instead, applying an RBAC-like model into MAS shifts the focus on runtime aspects, making roles, sessions, and policies the key runtime issues of a MAS organisation. In particular:

- RBAC users are represented in RBAC-MAS by *Agent Classes*;
- the behaviour of each role (agent) is defined in RBAC-MAS in terms of *Actions* and *Perceptions* used, respectively, to affect and perceive the computational environment where the agent is situated;

– *Policies* constrain the admissible interaction histories of an agent playing a specific role, and are used to explicitly model the organisational rules: at runtime, they are enforced by the underlying MAS infrastructure;
– while RBAC equips agents with a default role-set, in RBAC-MAS the agents' session starts with no activated roles: roles can be subsequently activated on a step-by-step basis, according to the specified activation/deactivation policies. The dynamics of role activation is constrained by the DSD rules.

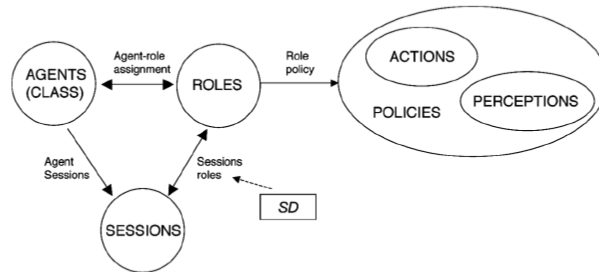

Figure 1. RBAC-MAS reference model [9]

More generally, RBAC approaches split some security issues which were previously faced by individual applications between the design and the infrastructure levels. Analogously, the factorisation in terms of agents of some security issues that frequently emerge in the engineering of a distributed system involves both the extension of MAS infrastructures with suitable services, and the improvement of the methodological support towards the design of complex secure systems.

2.2 Requirements for Engineering an RBAC-MAS System

In order to support the analysis and design of an access control system, a methodology should properly support the analysis and design of all the abstract entities adopted by the access control technique of choice.

In the specific case of RBAC-MAS, a methodology should allow engineers to model and design roles, organisations, objects, policies, operations, as well as static and dynamic constraints. Although at a first sight this requirement might seem somehow obvious, and possibly even superficial, a more detailed analysis leads to highlight some interesting aspects, that we will discuss below. Moreover, the separation between policy and mechanism introduces further constraints: in fact, while such two sub-systems can be designed separately, they are indirectly coupled by the representation language of the access policies, since these are designed by one sub-system, but enforced by the other. So, while it is not necessary to know the specific policy during the mechanism design phase, knowing how

the policy is represented is relevant to choose the most appropriate storage and to decide the most adequate enforcing implementation.

It is important to note that in the original RBAC-MAS formulation operations and objects collapse to the same entity “action” [14], i.e., an operation over a given object. However in the following we prefer to characterise both actions – the operation over the object – and objects, given that, from a software engineering viewpoint, the object entity hides all the environment abstractions and structures. The knowledge of the environment and of its topological structure is instead crucial when we deal with the engineering of a new system specially in the context of MAS engineering where is often necessary to design new environment abstractions in order to support the achievement of the agent’s tasks/goals. Obviously, the same distinction is not so crucial in the context of MAS infrastructures – where RBAC-MAS was developed – because there environment is already analysed and designed in terms of the infrastructures themselves. Finally, the distinction between actions and objects is very important also in the engineering of the interactions between agents and environment.

In turn, abstract entities add their own requirements, which can be outlined as follows:

Role — This entity implies that the methodology should support the modelling and design of both the user roles and the administrative roles which are required for the system management.

Organisation — This entity implies that the methodology should support the modelling and design of agent societies and of the rules that govern them.

Object — This entity hides a lot of complexity: in fact, the ability of modelling the “system’s objects”(i.e., the system “resources”) requires that the methodology is able to model the environment of the MAS. In the case of the mechanism sub-system, this means that the methodology should be able to both model and design the environment, since the mechanism has to provide the physical and logical control to prevent unauthorised access. In addition, the notion of MAS environment, as suggested in [16], implies both the modelling of the environment abstractions – entities of the environment encapsulating some functions – and of the topology abstractions – entities of MAS environment that represent the (either logical or physical) spatial structure. In fact, enabling system requirements to determine the topological structure of the system is necessary in order to capture the wide range of access control systems (e.g., from controlling the access to a file, to a room in a building, etc). Summing up, topology constraints should be considered since the earliest requirement analysis phase. So, supporting the object entity requires that a methodology enables engineers to model and design both the topological structure of the environment and the resources that populate it.

Action and Perception — These entities imply that the methodology should support the modelling and design of the actions that roles can perform over the objects and of the perceptions of the environment—as highlighted in the Subsection 2.1.

Policy — This entity leads to the design of rules concerning the abstractions. More precisely, these policies represent rules that involve roles, objects and actions, and rules over role activations and default roleset.

Finally, the mechanism sub-system should manage the association between users and roles, and should do that in a dynamic way: indeed, policies could change over time and at any time, and the sub-system should be able to support and implement such changes with no need to be stopped or reset.

3 SODA and RBAC-MAS

3.1 The SODA Methodology

SODA (Societies in Open and Distributed Agent spaces) [17,18] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts meta-model (A&A) [19], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process [20].

The SODA *abstractions* – explained below – are logically divided in three categories: *i)* the abstractions for modelling/designing the system active part (task, role, agent, etc.); *ii)* the abstractions for the reactive part (function, resource, artifact, etc.); and *iii)* the abstractions for interaction and organisational rules (relation, dependency, interaction, rule, etc.). In its turn, the SODA *process* is organised in two phases, each structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories – that is, at least one abstraction for the system active part, one for the reactive part, and another for interaction and organisational rules. Figure 2 shows an overview of the methodology structure: as we will show in the case study (Section 4), each step is practically described as a set of relational tables (listed in Figure 2 for the sake of completeness).

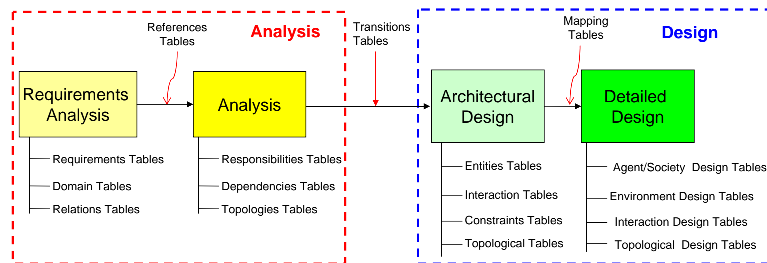

Figure 2. An overview of the SODA process

Requirements Analysis. Several abstract entities are introduced for requirement modelling. In particular, *requirement* and *actor* are used for modelling the customers' requirements and the requirement sources, respectively, while the *external-environment* notion is used as a container of the *legacy-systems* that represent the legacy resources of the environment. The relationships between requirements and legacy systems are modelled in terms of a suitable *relation*.

Analysis. The Analysis step expresses the requirement representation in terms of more concrete entities such as *tasks* and *functions*. Tasks are activities requiring one or more competences, while functions are reactive activities aimed at supporting tasks. The relations highlighted in the previous step are here the starting point for the definition of *dependencies* among such abstract entities. The structure of the environment is also modelled in terms of *topologies*, i.e., topological constraints over the environment.

Architectural Design. The main goal of this stage is to assign responsibilities of achieving tasks to *roles*, and responsibilities of providing functions to *resources*. In order to attain one or more tasks, a role should be able to perform *actions*; analogously, the resource should be able to execute *operations* providing one or more functions. The dependencies identified in the previous phase become here *interactions* and *rules*. Interactions represent the acts of the interaction among roles, among resources and between roles and resources, while rules enable and bound the entities' behaviour. Finally, the topology constraints lead to the definition of *spaces*, i.e., conceptual places structuring the environment.

Detailed Design. Detailed Design is expressed in terms of *agents*, agent *societies*, *composition*, *artifacts* and *aggregates* and *workspaces* for the abstract entities, while the interactions are expressed by means of *uses*, *manifests*, *speaks to* and *links to* concepts. More precisely, agents are intended here autonomous entities able to play several roles, while a society can be seen as a group of interacting agents and artifacts when its overall behaviour is essentially an autonomous, proactive one. The resources identified in the previous step are here mapped onto suitable artifacts, while aggregates are defined as a group of interacting agents and artifacts when its overall behaviour is essentially a functional, reactive one. The *workspaces* take here the form of an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces.

3.2 RBAC Requirements in SODA

In Subsection 2.2, two major requirement categories were outlined: one about the representation language of the access policies, the other concerning the RBAC-MAS abstract entities. With respect to the former issue, SODA is conceptually orthogonal to any possible representation language, since at the moment no hypothesis is made about the language adopted for compiling its relational tables.

With respect to the latter issue, RBAC-MAS abstract entities are captured by suitable SODA abstractions as follows:

Role — this RBAC-MAS entity is directly mapped onto SODA *role* concept in the Architectural Design phase, namely in the Entities Tables.[1]

Organisation — this entity is mapped by SODA *societies* in the Detailed Design phase (see the homonymous tables); in turn, the rules governing such societies are embodied in the social artifacts expressed by the Environment Design Tables.

Object — since this RBAC-MAS entity is used to represent the environment of the system, and given that topology constraints need to be considered since the earliest requirement analysis phase, its impact spreads through all SODA phases. More precisely, since the environment is composed of both environment abstractions and topological abstractions, in the analysis phase such aspects are captured via the *legacy system* and *function* abstractions, and the *topology* abstractions, respectively. Then, in the Design phase (which is particularly relevant for the mechanism sub-system), environment abstractions take the form of *resources* (in the Architectural Design step) and *artifacts* (in the Detailed Design step); in the same way, topology abstractions take the form of *spaces* (in Architectural Design) and *workspaces* (in Detailed Design).

Action and Perception — these entities are natively supported by SODA: *actions* and *use* map action in the Architectural Design and in the Detailed Design respectively, while *manifest* maps perception in the Detailed Design.

Policy — this entity finds it specific counterpart in the *Rule* abstraction (Constraints Tables) of SODA's Architectural Design, since in the Analysis phase they are simply considered as relations and dependencies; such rules are then mapped onto suitable (individual or social) *artifacts* in the Detailed Design step.

Considering how SODA supports RBAC-MAS requirements with respect to the three abstractions categories outlined above in this Section, it is worth noting that in the design of the mechanism sub-system only the reactive abstractions are involved, while in the design of the policy sub-system only the interactions and rules abstractions are used: the active abstractions, instead, are just modelled – not designed –, as in this kind of system the corresponding roles, from the RBAC design perspective, are just an input of the system, defined in the policy sub-system requirements. This is no longer true if a new system is being designed from scratch, where it is unlikely to have such roles as inputs of the whole system: rather, in such cases these roles should likely be first designed—and only then used as inputs in the design of the policy sub-system.

---

[1] The term "Entities" in SODA tables is used with a different, and more specific, semantics than in RBAC-MAS: Entities Tables, in fact, refer to roles, resources, actions, and operations, while RBAC-MAS uses that term to refer to a wide range of abstractions—from roles to organisations, objects, operations, permissions and constraints.

Summing up, all the key RBAC-MAS issues discussed above are quite well captured in SODA: such aspects are naturally taken into account when the methodology is applied, and take the form of suitable requirements and specifications in the SODA tables, as we will show in the case study below.

4 The Case Study

The case study we consider is the management of the access control to a university building: for the sake of brevity, we set the core layer quite at a high abstraction level, and only a limited set of the SODA tables will be reported. As highlighted in Section 2, the design of mechanisms (Subsection 4.1) is kept separate from the design of policies (Subsection 4.2). Here we present the key system aspect, that is, the topological structure of the environment, leaving the discussion of the application roles involved to the sub-systems design.

The topological structure, shared between the sub-systems, is established during the requirements analysis, since it derives from the physical structure of the building. More precisely, following the hierarchial view in Figure 3, layer *a* represents the whole building, layer *b* represents the spatial organisation of the building in terms of classrooms, administration, departments, and library, layer *c* represents the spatial organisation inside each department – composed of administration offices, professors' offices and the library – and administration – made of offices; finally, layer *d* represents the spatial organisation inside the administration department, which is, again, made of offices. Such a hierarchical representation simplifies the design of both mechanisms and policies, since the same mechanism can be replicated in the access points of the building, while finer-grained policies can be expressed for each space.

The general description above summarises the analysis of the topological constraints for both the mechanism and policies. As far as the mechanisms are concerned, this analysis suggests that the designer organises the environment following the topological structure, nesting spaces and workspaces, and mapping spaces onto workspaces: this approach simplifies the design of the mechanisms, which can be structured along different control levels. On the other hand, the

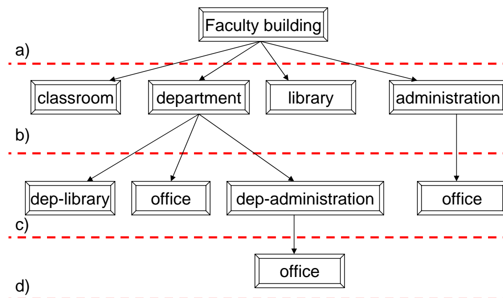

Figure 3. The physical structure of the university building.

policy designer can express accurate policies, defining whether each role can either access or not the building, along with its access privileges, for each access point—i.e., on a fine-grained basis.

4.1 Designing the Mechanism

The topological structure of the environment explained above is captured by two different SODA tables in the Architectural Design phase (Figure 4): the Space table ($(L)S_t$) describes the logical space of the system, while the Space-Connection table ($(L)SC_t$) shows the relation between spaces. Then, the workspaces in the Detailed Design step naturally follow from the spaces defined in the Architectural Design step.

Space	Description
Faculty	the whole building
Classroom	the student space
Library	the faculty library
Department	the research centre
Administration	the faculty bureaucracy centre
Dep-Library	the department library
Dep-Administration	the department bureaucracy centre
Office	the rooms for employees

Space	Connection
Faculty	Classroom, Library, Department, Administration
Administration	Office
Department	Dep-Library, Dep-Administration, Office
Dep-Administration	Office

Figure 4. Topological Structure in top down order: $(L)S_t$, $(L)SC_t$

According to this topological structure, the global mechanism can be conceived as composed of two complementary sub-mechanisms, one for the access to the whole building (Figure 5, top) and another for the access to a single room/office department—simply "room" in the following (Figure 5, bottom). Both are based on "Interface Artifacts" that represent the wrappers to the hardware resources capturing the user credentials: in the case of the whole building, there is an Interface Artifact for each hardware device that monitors a specific physical access point, while each room has its own Interface Artifact.

We assume that Interface Artifacts generate an event whenever a user enters the building (room); we also assume that such events are perceived by a suitable "(Room-)Access Manager" agent, whose task is to check whether such an access can be authorised. For this purpose, the Access Manager exploits the "User(-room) Artifact" to check if the user can access the building (room) and, if so, modifies the state of the "Building-State Artifact" accordingly. Room access,

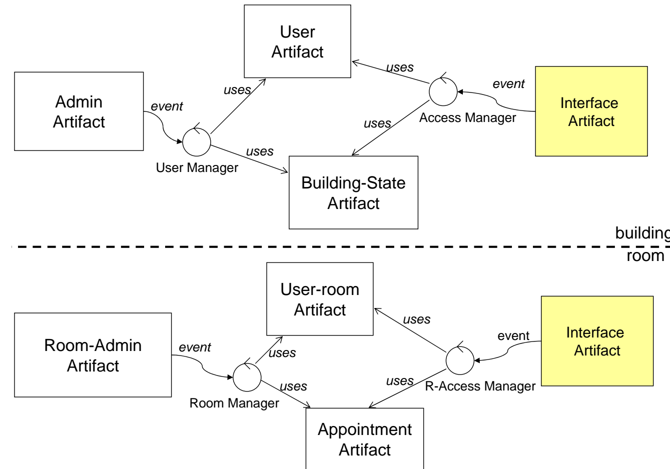

Figure 5. The mechanisms for the access control for the whole building (top) and the single room (bottom)

instead, must be granted also to users that are not permanently authorised, provided that they have an appointment: this is performed via the "Appointment Artifact".

The "User(-room) Artifact" stores all the roles permanently qualified to access the building (room), along with their access privileges. This artifact provides two different sets of functionalities: those needed to check the access authorisations, and those required by special users—such as administrators – for management purposes – such as adding or deleting roles, modifying the users' privileges, and so on. The "Building-State Artifact" traces the people inside the building: when a user exits the building, an event is generated, and the Access Manager modifies the artifact again. The "Appointment Artifact" manages the users' appointments, storing the list of the appointments for a given room: so, it is obviously shared by all the people that work in that room. The stored data include the time and the people involved in each appointment, enabling the policies designer to express several different policies—for instance, deciding whether the access should be denied or authorised if the people involved are not in office.

Users are managed by the "User Manager" agent, while users authorised to enter a room are managed by the "Room Manager": both such managers perceive the events generated by the "Admin Artifact" (respectively, by the "Room-Admin Artifact"). In turn, this represents the interface between the human administrator and the mechanism itself, and is used by the system administrator to introduce or delete roles and, more generally, to edit the policies over time (in the case of the building) or to handle appointments (in the case of rooms).

These functionalities are represented in the Artifact-UsageInterface table (AUI_t) of the Detailed Design (Figure 6): the usage interface represents the

set of operations provided by an artifact. For space reason, only the names of the artifacts' operations have been reported, by omitting operations' parameters.

Artifact	Usage Interface
Interface Artifact	enter_role, exit_role
Admin Artifact	new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
User Artifact	check_access, new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
Building-State Artifact	update_in, update_exit, get_roles
Room-Admin Artifact	new_role, canc_role, update_role, get_roles, new_user,canc_user, update_user, can_pol, new_pol, update_pol, insert_appointment, modify_appointment, delete_appointment, get_appointments
User-room Artifact	check_access, new_role, canc_role, update_role, get_roles, new_user, canc_user, update_user, can_pol, new_pol, update_pol
Appointment Artifact	check_appointment, insert_appointment, modify_appointment delete_appointment, get_appointments

Figure 6. Artifact-UsageInterface table

4.2 Designing RBAC Policies

RBAC policies are designed during SODA's architectural design phase: more precisely, the constraints that shape the role interaction spaces drive the design of the organisational rules.

In our case, the environment needs not – and actually can not – be explicitly designed, as it is already represented in/by the above mechanism as specified in Subsection 3.2: all we need is to model it in the analysis phase, so as to identify the relationships and the interactions between the two sub-systems. Then, the mechanism's artifacts will enforce the policies designed here, while, conversely, the roles (agents) defined here will interact with the mechanism. For the same reason, also the topological structure is implicit in the mechanism: so, the spaces/ workspaces identified in Subsection 4.1 are the same here, too. As a result, we now focus only on the design of the interaction and organisational rule entities.

From the viewpoint of sub-system requirements, our scenario, in its simplest version, involves six different roles: *Student*, *Professor*, *Technician*, *Administrative staff*, *Guide* and *Visitor*. Professors, Technicians, and Administrative staff can freely access the building at any time. Students, instead, can access the building – in particular, classrooms and library – only during the regular opening hours; in addition, to access the Administrative staffs' and Professors' offices, they must have an appointment. Finally, Visitors cannot access the building without a Guide, who is a member of the University – Professor, Technician, Administrative staff – that escorts visitors inside the building.

Beyond these roles, the user management activity highlights the need of a new "service" role – the *System administrator* – for modifying the access privileges and managing the users' credentials. This is not surprising, since during SODA's

Role	Action
Visitor	enter, exit, ask_appointment
Student	enter, exit, ask_appointment
Professor	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Administrative staff	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Technician	enter, exit, canc_appointment, set_appointment, change_policy insert_role, canc_role
Guide	enter, exit
System administrator	enter, exit, change_policy, insert_role, canc_role

Figure 7. Role-Action Table ($(L)RA_t$)

Architectural Design new roles are often identified that complete and support the activities of the roles directly deducted from the requirements.

So, there are seven different roles in all, each potentially able to perform the actions depicted in Figure 7—we say "potentially" because the role will actually be enabled to do such actions only if/when authorised to. Rules derive from the desired policies, and are listed in Figure 8: the corresponding association to roles is given in Figure 9.

For the sake of clarity, Figure 8 is organised in different sets of rules. The first set (Guide-Rule and Visitor-Rule) reports the DSD and SSD constraints over the corresponding roles (shown in Figure 9) that are enforced by the "User Artifact". In particular, the *Guide* role is dynamically incompatible with any other role during a session (DSD constraint), since the Guide cannot abandon visitors, who are not allowed to move alone inside the building. Similarly, the *Visitor* role is incompatible with any other because a visitor cannot cover any position inside the university: this is an SSD constraint, since this incompatibility holds permanently (it is not related to a temporary status in the session). The second set of rules represents the constraints over the administrative operations, enforced by the the "(Room-)Admin Artifact". The two other sets express, respectively, the constraints over the access to the building (third set) and to each room (fourth set), and are enforced, respectively, by the "User Artifact" and by the "User-room Artifact" & "Appointment Artifact" pair.

5 Conclusions and Related Work

To the best of our knowledge, this is the first attempt to support the design of an RBAC system via an agent-oriented methodology such as SODA. In fact, other works in the literature (e.g. [21,22]) exploit MAS for realising an RBAC system, but in the context of specific domain applications: thus, they lead to ad-hoc solutions which are not easily reusable in other contexts, due to the lack of separation between the "static part" of the system – the mechanism – and the "dynamic part" – the policies. In addition, these works delegate the enforcing

Rule	Description
Guide-Rule	Guide cannot be activated together other roles (DSD constraint)
Visitor-Rule	Visitor cannot be activated together other roles (SSD constraint)
Admin-Rule	The Administrator can modify the access rules for the whole building but cannot modify the access rules for the offices
Prof-Admin-Rule	The Professor can modify the access rules for his/her office
Staff-Admin-Rule	The Administrative staff can modify the access rules for their office
Visit-Rule	Visitor can access the building only together a Guide
Building-Rule	The access to the building is possible only when the building is open to the public
Uni-Build-Rule	Professor, Technician, Administrative staff and System administrator can always access the building
App-Rule	The access to an office is granted only if the Student has an an appointment and the Professor/Administrative staff is in the office
Administration-Rule	The access to the staff office is possible only when the office is open to the public
ClassRoom-Rule	The access to a classroom is not granted during a lecture
Library-Rule	The access to the library is possible only when the library is open to the public
Lab-Rule	The access to the laboratory is possible only when the laboratory is open to the public
Department-Rule	The access to the department is possible only if the destination room grants the access

Figure 8. Rule table $((L)Ru_t)$

of policies to agents, while our approach is that such an enforcing should more properly be done by suitable environmental abstractions [9].

Moving from the "university building access" case study, this paper aims at showing the benefits of a clear separation between mechanism and policies, so as to split the design of an access control system in two separate aspects: our SODA-based approach leads to design such aspects as two sub-systems, exploiting the agent paradigm. In particular, the mechanism sub-system is designed as general as possible, since its structure is basically stable and reusable as is in other applications: artifacts wrap the physical resources, and a society of agents reacts to the events occurring in the environment. From this viewpoint, SODA's intrinsic support for both environmental abstractions – artifacts – and topology abstractions – workspaces – makes it possible to support the whole design process of the environment, including its spatial structure, in a uniform way.

Policies, on the other hand, are generally tied to the application domain, so they typically have to be re-designed each time: as highlighted in Subsection 4.2, the design of this sub-system is focused on the definition of roles and their access privileges, which are the core of any RBAC system. Again, SODA natively supports both roles and access privileges, which can be easily expressed in terms of rules. So, only one sub-system needs to be redesigned in response to any application or policy change—the other sub-system remains untouched, unlike what would happen with a monolithic system.

Role	Rule
Visitor	Visitor-Rule, Visit-Rule, Building-Rule, Administration-Rule ClassRoom-Rule, Library-Rule, Department-Rule
Student	Building-Rule, App-Rule, Administration-Rule, Lab-Rule ClassRoom-Rule, Library-Rule, Department-Rule
Professor	Uni-Build-Rule, Administration-Rule, Lab-Rule Library-Rule, Department-Rule, Prof-Admin-Rule
Administrative staff	Uni-Build-Rule, Administration-Rule, Lab-Rule Library-Rule, Department-Rule, Staff-Admin-Rule
Technician	Uni-Build-Rule, Administration-Rule,Library-Rule, Department-Rule
Guide	Guide-Rule, Uni-Build-Rule, Administration-Rule, Lab-Rule ClassRoom-Rule, Library-Rule, Department-Rule
System administrator	Uni-Build-Rule, Administration-Rule, Admin-Rule Library-Rule, Department-Rule

Figure 9. Role/Rule association table $(L)RoRu_t$

Future work will be mainly devoted to improve the methodology in several directions: *i)* to support the design of secure agent-oriented systems since the earliest Requirement Analysis step; *ii)* to develop a language for SODA rules which could be able to capture all the relevant RBAC permissions and constraints, and *iii)* to more deeply study the access control issues related to artifacts.

References

1. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems. 2nd edn. Wiley Computer (2001)
2. Mouratidis, H., Giorgini, P.: Secure tropos: A security-oriented extension of the tropos methodology. International Journal of Software Engineering and Knowledge Engineering **17** (2007) 285–309
3. Henderson-Sellers, B., Giorgini, P., eds.: Agent Oriented Methodologies. Idea Group Publishing, Hershey, PA, USA (2005)
4. Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers (2004)
5. Bernon, C., Cossentino, M., Pavón, J.: An overview of current trends in european AOSE research. Informatica **29** (2005) 379–390
6. Liu, L., Yu, E., Mylopoulos, J.: Analyzing security requirements as relationships among strategic actors. In: 2nd Symposium on Requirements Engineering for Information Security (SREIS'02), Raleigh, NC, USA (2002) Electronic Proceedings.
7. Yu, E., Cysneiros, L.M.: Designing for privacy and other competing requirements. In: 2nd Symposium on Requirements Engineering for Information Security (SREISí02), Raleigh, NC, USA (2002) Electronic Proceedings.
8. Omicini, A., Ricci, A., Viroli, M.: RBAC for organisation and security in an agent coordination infrastructure. Electronic Notes in Theoretical Computer Science **128** (2005) 65–85 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04), 30 August 2004. Proceedings.
9. Viroli, M., Omicini, A., Ricci, A.: Infrastructure for RBAC-MAS: An approach based on Agent Coordination Contexts. Applied Artificial Intelligence **21** (2007) 443–467 Special Issue: State of Applications in AI Research from AI*IA 2005.

10. Johnson, M., Feltovich, P.J., Bradshaw, J.M., Bunch, L.: Human-robot coordination through dynamic regulation. In: IEEE International Conference on Robotics and Automation, IEEE Computer Society (2008) 2159–2164 2008 IEEE International Conference on Robotics and Automation ICRA 2008 Pasadena, California, on May 19–23, 2008.
11. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models. IEEE Computer **29** (1996) 38–47
12. RBAC: American National Standard 359-2004 (Role Base Access Control – home page). `http://csrc.nist.gov/rbac/` (2004)
13. Ferraiolo, D., Kuhn, R., Sandhu, R.: RBAC standard rationale: Comments on a critique of the ANSI standard on Role Based Access Control. IEEE Security & Privacy **5** (2007) 51–53
14. Omicini, A., Ricci, A., Viroli, M.: An algebraic approach for modelling organisation, roles and contexts in MAS. Applicable Algebra in Engineering, Communication and Computing **16** (2005) 151–178 Special Issue: Process Algebras and Multi-Agent Systems.
15. Ricci, A., Viroli, M., Omicini, A.: An RBAC approach for securing access control in a MAS coordination infrastructure. In Barley, M., Massacci, F., Mouratidis, H., Scerri, P., eds.: 1st International Workshop "Safety and Security in MultiAgent Systems" (SASEMAS 2004), AAMAS 2004, New York, USA (2004) 110–124 Proceedings.
16. Molesini, A., Omicini, A., Viroli, M.: Environment in Agent-Oriented Software Engineering methodologies. Multiagent and Grid Systems **4** (2008) Special Issue on Environment Engineering for MAS.
17. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A roadmap to artefacts. In Dikenelli, O., Gleizes, M.P., Ricci, A., eds.: Engineering Societies in the Agents World VI. Volume 3963 of LNAI. Springer (2006) 49–62 6th Inter. Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 October 2005. Revised Paper.
18. SODA: Home page. `http://soda.apice.unibo.it/` (2008)
19. Omicini, A.: Formal ReSpecT in the A&A perspective. Electronic Notes in Theoretical Computer Sciences **175** (2007) 97–117 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.
20. Molesini, A., Omicini, A., Ricci, A., Denti, E.: Zooming multi-agent systems. In Müller, J.P., Zambonelli, F., eds.: Agent-Oriented Software Engineering VI. Volume 3950 of LNCS. Springer (2006) 81–93 6th Inter. Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.
21. Drouineaud, M., Lüder, A., Sohr, K.: A role based access control model for agent based control systems. In Unland, R., Ulieru, M., Weaver, A.C., eds.: 1st IEEE International Conference on Industrial Informatics (INDIN 2003), Banff, Alberta, Canada (2003) 307–311
22. Yamazaki, W., Hiraishi, H., Mizoguchi, F.: Designing an agent-based RBAC system for dynamic security policy. In: IEEE 13th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2004), 9th International Workshop "Enterprise Security" (ES 2004), Modena, Italy, IEEE Computer Society (2004) 199–204