Multi-OrBAC: a New Access Control Model for Distributed, Heterogeneous and Collaborative Systems

Anas Abou El Kalam (LIFO / ENSIB), Yves Deswarte (LAAS–CNRS / Université de Toulouse)

anas.abouelkalam@ensi-bourges.fr, yves.deswarte@laas.fr

***Abstract*—This paper presents a new access control model for collaborative, heterogeneous and distributed systems: the *MultiOrganization-Based Access Control* (Multi-OrBAC). Multi-OrBAC layers heterogeneity and collaboration access control concepts on top of the OrBAC model. Moreover, Multi-OrBAC is represented by UML diagrams and associated to a formal system based on Logical Programming by Constraint. Finally, we specify a Multi-OrBAC driven architecture and we present a use case to describe how the relevant steps related to security (e.g., authorization, credentials and proofs generation and verification) are implemented.**

***Index Terms*—Access control, security policies and models, heterogeneous and collaborative system security, conflict detection and resolution, XACML.**

I. Introduction

Doing business (e.g. by building a “community of interest”) in today’s world of distributed systems usually requires collaboration between different organizations. Consequently, developing access control policies and models for such cross-border co-operations is an important issue.

Before studying this issue, let us explain some basic concepts. The security policy is defined by the Common Criteria such as *the set of laws and rules regulate how an organization manages, protects, and distributes sensitive information* [1]. Nevertheless, the security policy can be badly designed and intentionally or accidentally violated. A security model is intended to: abstract the policy and handle its complexity; represent the secure states of a system as well as the way in which the system may evolve, verify the consistency of the security policy and detect possible conflicts.

OrBAC Model helps organizations in defining their policies [2]. However, OrBAC have several weaknesses and is not really adapted to collaborative systems. In this work we extend OrBAC to the Multi-OrBAC model. Our major aim is to overcome the weaknesses of OrBAC and to address access control and authorization problems in large-scale, decentralized systems. Such problems arise, for example, when independent organizations enter into co-operations.

While sharing resources, each organization keep authority over the resources it controlled prior to entering the co-operation. Such systems are called multicentric collaborative systems, since they have no single central authority [3].

We particularly emphasis on the following points:

- use the Unified Modeling Language [4] to represent the basic concepts of the Multi-OrBAC policy;
- use the logical programming by constraints to model the operational rules of the system and to formalize the security policy;
- provide resolution and inheritance deduction engines;
- present a global architecture and an implementation (logic, Java, XACML).

The remainder of this paper is organized as follows: Section 2 and 3 discuss the traditional security models and OrBAC. Then, Section 4 describes the concepts of MultiOrganization-Based Access Control. Section 5 specifies the Multi-OrBAC formal system as well as the conflict resolution mechanism. Afterwards, through a case study, Section 6 gives details of the authorization phase and presents a Multi-OrBAC driven architecture. Section 7 compares Multi-OrBAC with Role-Based Trust Management (RBTM) and integrates Multi-OrBAC in a XACML implementation. Finally, in Section 8 we present our main conclusions and discuss future works.

Note that due to space shortage, neither the administration model nor the negotiation process will be presented in this contribution.

II. Traditional Security Models

Classical access control policies and models (discretionary “DAC” and mandatory access control “MAC” [5]) are not really adapted to distributed and heterogeneous systems. For instance, HRU represents the relationships between the subjects, the objects and the actions by a matrix M [6]. $M(s, o)$ represents the actions a that a subject s is allowed to carry out on o. It is thus necessary to enumerate all the triples (s, o, a) that correspond to permissions defined by the security policy. Moreover, when new subjects, objects or actions are added or removed in the system, it is necessary to update the policy.

Role Based-Access Control (RBAC) is more flexible. Roles are assigned to users, permissions are assigned to roles and users acquire permissions by playing roles [7, 8]. Hierarchical RBAC [9] adds a requirement for supporting the role hierarchies, while constrained RBAC [10] enforces the separation of duties. RBAC is unquestionably suitable for a large range of organizations. Indeed, if users are added to the system, only the instances of the relationship between the users and the roles are updated. Besides, RBAC is deliberately policy neutral. In applying RBAC to a system, the interpretation of permissions is an important step to perform.

The OrBAC (Organization-based Access Control) model is an extension of RBAC that details permissions while remaining implementation independent. The main idea is to express the security policy with abstract entities only, and thus to completely separate the representation of the security policy from its implementation. Indeed, OrBAC is based on roles, views, activities (introduced in RBAC, VBAC, TBAC [11, 12]) to structure the subject, the objects and the activities.

In the next Section, we first summarize OrBAC and we discuss the limits of this model. Afterwards, in Section 4, we extend OrBAC to overcome these limits and to cover the particularities of collaborative and multicentric systems.

III. ORBAC (ORGANIZATION-BASED ACCESS CONTROL)

In OrBAC, the entity Role is used to structure the link between the subjects and the organizations (Fig.1). The relationship *Employ(org, r, s)* of the means that org employs subject s in role r. In the same way, the objects that satisfy a common property are specified through views (Fig. 2), and activities are used to abstract actions (Fig. 3).

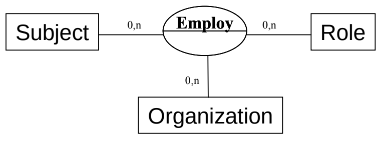


Fig1. Abstracting subjects.

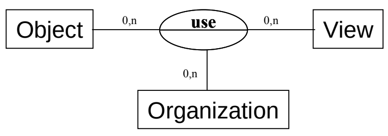


Fig2. Abstracting objects.

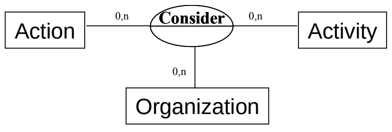


Fig. 3: Abstracting actions.

Security rules have the following form *Permission(org; r; v; a, c)*: in the context c, organization org grants role r the permission to perform activity a on view v.

OrBAC improves the management of the security policy and reduce considerably its complexity. However, it is partly limited.

Firstly, OrBAC is only represented with an entity-relation diagram. This kind of representation is not rich enough; in particular, it does not capture some of the conceptual (e.g., aggregation, dependence, hierarchy) and dynamical aspects of access control.

Secondly, the translation of the security policy to access control mechanisms is not treated in OrBAC. It seems necessary to describe the architecture and the implementation (e.g., credential's generation).

Thirdly, the multiplicity used in the OrBAC relationships (Fig. 1, 2, 3) is incorrect. In Fig. 2, for example, the *0,n* multiplicity at the object side means that some views can be empty (no object is belonging to the view); the *0,n* multiplicity at the view side means that some objects do not belong to any view; and the *0,n* multiplicity at the organization side means that some organizations do not contain any object.

Fourthly, OrBAC is not adapted to heterogeneous, distributed and interoperable systems. Let us examine the following questions:

- is OrBAC able to represent security rules that involve several organizations? In particular, is it possible to grant or deny access to users belonging to other external organizations?
- in a particular organization, it is possible to have several views with heterogeneous implementations?

Concerning the first question, as security rules have the *Permission(org; r; v; a, c)* form, it is not possible to represent rules that involve several organizations. Indeed, a single organization is specified in each rule. OrBAC is only adapted to centralized structures.

Concerning the second question, as the actions depend only on the organizations and the activities (Fig. 3), and do not take the "view" into account, it is not possible to have objects on which we can carry out different actions. In fact, if we assume that an organization has the views *medical record*, *financial records* and *printers*, the action that we can carry out on the three views must be the same; this restriction implies that all the objects of a particular organization have the same format!

To overcome these limits, we extend OrBAC. We first describe the main concepts of Multi-OrBAC. Then, in Section 5, we suggest a Multi-OrBAC formal model. In Section 6, we derive a Multi-OrBAC driven architecture, as well as a case study and we describe our implementation.

IV. MULTI-ORBAC

A. *Role in organization*

In some cases, even if a user is allowed to play several roles, it does not necessarily have the right to play them in any organization. For example, David may be authorized to play the "interventional radiologist at *hospital*$_1$" and "radiologist at *hospital*$_2$" roles, but he may not be allowed to play the interventional radiologist at *hospital*$_1$, nor the radiologist at *hospital*$_2$.

As it is not sufficient to associate users to roles, Multi-OrBAC adds the new "*Role in Organization*" (R*i*O) entity, and associates subjects to R*i*O (Fig. 4). Obviously, a particular user can play several roles in the same organization.

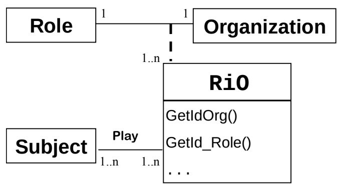


Fig. 4: Role in Organization.

The R*i*O has the characteristics of a class and an association; it is modelled by a class-association. This means that when an instance of a role is associated with an instance of an organization, there will also be an instance of a *RiO*.

A R*i*O is composed of one (and only one) role and one (and only one) organization; a role (or organization) can participate to one or several R*i*O; a subject can play one or several R*i*O; and conversely, one or several subjects can play a RiO.

B. View in organization

As views characterize the way objects are used in organizations, the class-association "View in Organization" (*ViO*) is thus introduced (Fig. 5). With *ViO*, it is possible to express that different organizations could implement the same view; for example, if *hospital*$_1$ uses XML files and if *hospital*$_2$ uses a database system, the *medical record* view corresponds to XML files in *hospital*$_1$, whereas in *hospital*$_2$, it corresponds to a table.

A V*i*O contains at least one object, and each object belongs to one or several V*i*O.

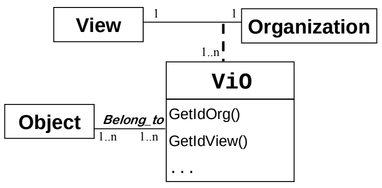


Fig. 5: View in Organization.

C. Activity in organization

Different organizations may decide that the same action corresponds to distinct activities, and similarly, the same activity could be implemented differently in accordance to organizations (Fig. 6). For example, the *consulting* activity corresponds, in *Org$_A$*, to an *open-Xml-File()* action whereas it corresponds to *select* action in *Org$_B$*.

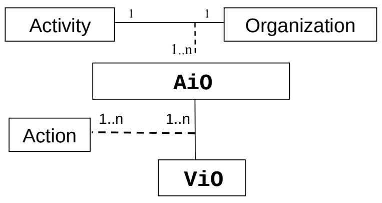


Fig. 6: Activity in Organization.

D. Context in Organization

Several definitions were given to the security context. The first one was "a *software that adapts according to its location of use, the collection of nearby people and objects, as well as the changes to those objects over time*" [13]. Covington et al. [14] expands the RBAC model by incorporating environment roles to capture environmental attributes; Bertino et al. [15] have examined temporal authorization in databases; etc. OrBAC states that the context should be taken into account; however, it does not give more details. In Multi-OrBAC we define the context as "*any constraint or information (location, time, physical or computational object, etc.) that helps to specify the concrete circumstances of the access request*". This definition is a little similar to the XACML context definition (in XACML, the context is an attribute of a resource, an object or an action).

In this work, we suggest proceeding in two steps:

Firstly, the context is specified by first-order or temporal predicates that identify the conditions when this context is true or not. For example:

- The context "Local_Access" is true if and only if "{@_Request in 192.192.1.* ∨ @_Request = 126.15.1.3}.
- The context "Night_Access" is true if and only if {time_request ∊ [6h pm, 7h am]}.
- The context "Attending_Physician(o, s)" is true between subject s and object o, if and only if Name(o) ∊ Patient (s).

Secondly, in order to provide a fine-grained access control, the context should figure in each rule; e.g., without using contexts, we can state that "physicians can access medical records"; however if we use the context, this security rule can be converted to "physicians can access medical records only if the context *Attending_Physician* is true".

In Multi-OrBAC, the "*Context in Organization*" (C*i*O) can be defined as a class association.

E. Security rules

Two levels can be distinguished in Multi-OrBAC: abstract and concrete levels (Fig. 7).

Abstract level: the security administrator defines security rules through abstract entities without worrying about how each organization implements these entities. A security rule is specified as follows: *Permission* (R*i*O, A*i*O, V*i*O, C*i*O). Recommendations, prohibitions and obligations are defined similarly.

Concrete level: when a user u_i, requests an access, the concerned rules are instantiated, the parameters are evaluated and concrete authorizations are granted (or not) to u_i.

Basically, it is sufficient to express security rules in the *Permission* (R*i*O, A*i*O, *view*, *context*) form, because the organization of the view, the activity and the context are similar. Thus, a security rule could be *Permission*(Physician-*in*- *Org$_A$*, Reading-in- *Org$_B$*, MedicalRecord, disaster); this means: *Org$_A$*'s physicians are allowed to consult (in *Org$_B$*) *Org$_B$*'s medical records in case of a disaster (in *Org$_B$*).

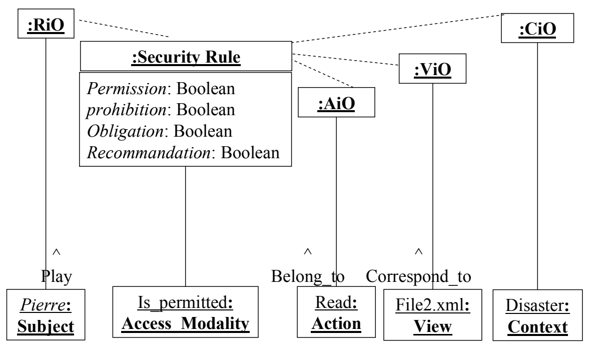


Fig. 7: Instantiating security rules in Multi-OrBAC

Fig. 8 presents the Multi-OrBAC model.

In this UML class diagram, each organization can be seen as a composition of several roles, activities and views on this organization (*RiO*, *AiO* and *ViO*). Furthermore, each organisation can contain several sub-organizations.

For more expressiveness and less complexity, we introduce role hierarchy: a role not only has its assigned permissions but also all the permissions of its sub-roles; likewise, the permissions defined for an organization are spread to its sub-organizations. The inheritance mechanism will be presented in Section 5.3.

Besides, subjects, activities and views have recursive structures. Let us take the example of the *Subject* entity: a user is a subject; a group of users (e.g., a team) is a subject; and a group is a set of subjects; e.g., the subject "employees of Org_Z" can contain $Team_1$, $Team_2$ and the supervisor *Bob*.

In the same way, an atomic activity is an activity, a composite activity is an activity, and an activity can contain one or several atomic activities and/or one or several composite activities.

Finally, it is important to remind that, unlike OrBAC that consider a central organization, Multi-OrBAC concerns large-scale, decentralized systems. Hence, each organization can define its own rules (for its own users and sub-organizations), but it also can negotiate foreign access with other organizations. Multi-OrBAC can thus be seen as a homogenous federation of several OrBAC's instantiations. The following hypothesis should be taken into account:

- organizations associate users to R*i*O, objects to V*i*O and actions to A*i*O; in this way, each organization has some functioning flexibility (e.g., its own services, file system, …) and can differently implement and instantiate views, activities and contexts;
- a prior mutual negotiation of rules that imply several organizations.

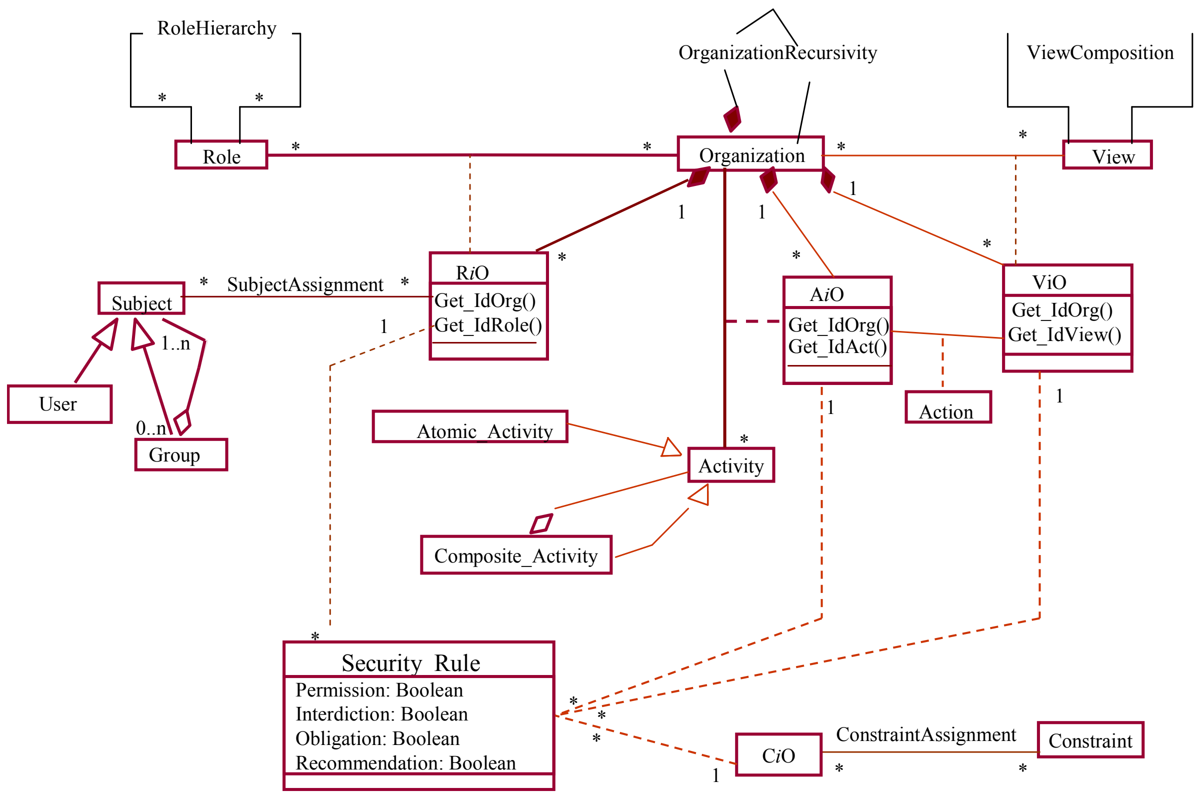


Fig. 8: The Multi-OrBAC model (UML class diagram).

V. MULTI-ORBAC FORMAL SYSTEM

A. The language choice

In this section, we suggest a model that is able to formally express Multi-OrBAC security policies, as well as operational rules, but also to manipulate the specification by using logical proofs, e.g, to:

- detect and resolve conflicting rules,
- derive permissions (e.g., by inheritance) and query a given security policy
- verify that properties (e.g., consistency and completeness) are enforced by the policy,
- verify that a given state (i.e., a situation) does not violate the security policy,
- investigate interoperability problems between several security policies, etc.

To deal with this kind of issues, we suggest using Logical Programming by Constraint (LPC). LPC is an extension of Logical Programming (LP) [18, 19].

LP is a programming paradigm based on a sub-set of the first-order logic, while LPC is particularly useful for detecting and resolving conflicts.

PL has several features that can be interesting for this study: its declarative presentation clarifies the description of the studied problem; its power (by using unification and resolution process) allows to easily infer the solution from the problem description; its flexibility makes it possible to use the same procedure to solve different problems (by changing the parameters of the procedure).

We used The Prolog language in our implementation of the conflicting detection and resolution engine. The latter is based on PL: it allows a declarative presentation of the problem and provides a resolution mechanism. Basically, a Prolog program is a set of clauses; a clause is an affirmation about logical atoms; an atom represents a relationship between terms; and terms are objects of the system [20].

B. Specification of a Multi-OrBAC policy

First, we define axioms related to permissions, obligations and recommendations. For example: Obligation*(RiO$_1$, AiO$_2$, V, C*) → Recommendation*(RiO$_1$, AiO$_2$, V, C*) "every obligation is a recommendation"; and *Recommendation(RiO$_1$, AiO$_2$, V, C)* → *Permission(RiO$_1$, AiO$_2$, V, C)*. These axioms will be useful in the conflict resolution process.

Secondly, the relationships of the Multi-OrBAC model (Fig. 7 and 8) are transformed into logical predicates. Indeed, we define predicates such as *sub_role*(*Role_junior*, *Role_senior*), *Play*(*subject, RiO*), etc. An instance of the first predicate could be *sub_role*(surgeon, physician) "a surgeon *is_a* physician", while an instance of the second predicate could be *Play*(David, Physician in *Org$_A$*) "David plays the R*i*O Physician in *Org$_A$*".

Thirdly, to derive permissions, we can use axioms of first order classical logic, but also axioms specific to Multi-OrBAC. For example, the derivation of permissions (described in Fig. 7 and 8) can be formally expressed as follows (Fig. 9):

Permission(*RiO$_1$*, *AiO$_2$*, *V*, *C*) ∧
Play (*s*, *RiO$_1$*) ∧
Correspond_to(*o*, *ViO$_2$*) ∧
Belong_to (α,*AiO$_2$*) ∧
Is_true(*CiO$_2$*)
→ ***Is_permitted*** (*s*, α, *o*).

Fig. 9: A Multi-OrBAC security rule.

An instance of this axiom could be
Permission(Physician-*in-Org$_A$*,
Reading-*in-Org$_B$*, MedicalRecord, disaster) ∧
Play(*Bob*, Physician-*in-Org$_A$*) ∧
Correspond_to(*f$_1$.xml*, MedicalRecord-in-*Org$_B$*) ∧
Belong_to (*Read-xml()*, Reading-in-*Org$_B$*) ∧
Is_true(disaster-*in-Org$_B$*)
→ ***Is_permitted*** (*Bob*, *Read-Xml-File()*, *f$_1$.xml*).

Basically, this axiom contains three elements:

- (The security rule) if within the context disaster-*in-Org$_B$*, Physician-*in- Org$_A$* has the permission to perform Reading-*in-Org$_B$* on MedicalRecord-*in-Org$_B$*,
- (The conditions) if *David* plays Physician-*in-Org$_A$*, if *f$_1$.xml* corresponds to MedicalRecord-in-*Org$_B$*, if *Read-XML-file()* belongs to the Reading-in-*Org$_B$ AiO* and if the context disaster-*in-Org$_B$* is true,
- (The effect) then *David* has permission to carry out *Read-XML-file()* on *f$_1$.xml*.

C. Inheritance rules specification

This section describes how we derive permissions by inheritance. Indeed, to minimize the complexity of the security policy, we consider inheritance relationships on roles, organizations, and activities. Let us present a few examples:

1. Role hierarchy: the security rule is as follows:
Permission(*Role$_{senior}$-in-Org$_A$*, *AiO*, *V*, *C*) ∧
sub_role(*Role$_{junior}$*, *Role$_{senior}$*)
→ ***Permission***(*Role$_{junior}$-in-Org$_a$*, *AiO*, *V*, *C*).

For instance,
Permission(*Physician-in-Org$_A$*, *AiO*, *V*, *C*) ∧
sub_role(*Surgeon, Physician*).
→ ***Permission***(*Surgeon-in-Org$_A$*, *AiO*, *V*, *C*).

If a subject plays *Surgeon-in-Org$_A$,* it has the permissions associated to *Surgeon-in-Org$_A$* but also the permissions assigned to *Physician-in-Org$_A$*.

2. We can also take advantage of the organization's composition. For example, the rule
Permission(*Role-in-Org$_1$*, *AiO*, *V*, *C*) ∧
sub_organization(*Org$_2$*, *Org$_1$*)
→ ***Permission*** (*Role in Org$_2$*, *AiO*, *V*, *C*).

Basically, if some permissions are granted to the "*physicians* a*t hospital$_1$*" *RiO* then we can infer that these permissions are granted to the physicians of all the sub-organizations of *hospital$_1$*, e.g., the radiology department and the emergency department of the *hospital$_1$* (in this example, *Role*=physician, *Org$_1$*=" *hospital$_1$*" and *Org$_2$*= "*Radiology department*, …).

3. Intuitively, we can also assume that if a role has permissions on objects of an organization *Org$_1$*, then he has these permissions on objects of the sub-organizations of *Org$_1$*. For example

Permission(Physician-*in- Org$_A$*,
Reading-*in- Org$_B$*, MedicalRecord, disaster) ∧
sub_organization(Department$_1$, *Org$_B$*)
→ *Permission*(Physician-*in-Org$_A$*, Reading-*in-Department$_1$*, MedicalRecord, disaster).

The inheritance rule is thus:

Permission(*RiO*, *Activity in Org$_{senior}$*, *View in Org$_{senior}$*, *Context in Org$_{senior}$*) ∧
sub_role(*Org$_{junior}$*, *Org$_{senior}$*)
→ ***Permission***(*RiO*, *Activity in Org$_{junior}$*, *View in Org$_{junior}$*, *Context in Org$_{junior}$*).

D. Conflicting resolution

Several definitions of the "policy conflict" have been given. The RFC3198 [21] states that a policy conflict *Occurs when the actions of two rules contradict each other.* Moffett [22] and Lupu [23] categorise conflicts into *modality conflicts* (contradiction between types of rules) and *application-specific conflicts* (due to model-specific inconsistencies or limitations). Strembeck has studied the problem of "conflict detection" in RBAC [24], Bertino et al. in databases [25], etc.

In this paper, we only deal with modality conflicts. In fact, we consider a conflicting situation when a user simultaneously has the:

- prohibition and the obligation to carry out the same action on a specific object, or
- prohibition and the permission to carry out the same action on a specific object, or
- prohibition and the recommendation to execute the same action on a specific object

Such a situation is possible as, at a given time, we could have several rules (permissions, prohibitions, obligations, recommendations) for the same (*RiO*, *AiO*, *v*, *c*). The conflict can occur:

- between existing security rules;
- between existing rules and a temporary intervention of the administrator, e.g., to add or modify a rule or an entity;
- between existing rules and a rule resulting from the application of the inheritance mechanism (spreading by inheritance).

To prevent conflicting situations, we should first verify the coherence of the security policy (*off-line verification*). In other words, we need to verify that the system is in a secure state. We should also make sure that every intervention of the security administrator keeps the system in a secure state (on*line verification*).

In this paper, we suggest using Logical Programming by Constraints (LPC). LPC is a combination of the *logical deduction process* and a set of *constraint resolving algorithms* [17, 18].

Basically, we affect priorities (an integer) to each security rule, e.g., ***Permission***(*RiO$_1$*, *AiO$_2$*, *V*, *C*, <u>*P*</u>): in the context *c*, R*i*O$_1$has the permission to perform *AiO$_2$ on* V*iO$_2$* with the *"P" priority.*

The access decision is deduced according to axiom of Fig. 10 (that replaces Fig. 9).

Permission(*RiO$_1$*, *AiO$_2$*, *V*, *C*, <u>*Priority*</u>) ∧
Play (*s*, *RiO$_1$*) ∧
Correspond_to(*o*, *ViO$_2$*) ∧
Belong_to (*α*,*AiO$_2$*) ∧
Is_true(*CiO$_2$*)
→ ***Is_permitted*** (*s*, *α*, *o*, <u>*Priority*</u>).

Fig. 10: Multi-OrBAC security axiom with priorities.

As for a particular (*RiO$_1$*, *AiO$_2$*, *V*, *C*) we can have different security rules (permissions, interdictions, …) with different priorities, it is possible to have conflicting decisions (with different priorities) for the same (*s*, *o*, *a*). To resolve these situations, we first assume that the system is in a secure state. Every intervention (e.g. to add a rule) of the administrator activates the following actions:

1. The new rule (e.g. *Permission(RiO$_1$, ViO$_2$, A, C, Priority$_a$)*) is first kept in a temporary base;
2. The authorization server invokes the process that extracts the relevant rules; this is achieved by extracting the rules that have the same attributes *RiO$_1$, ViO$_2$, A, C*, for example *Interdiction(RiO$_1$, ViO$_2$, A, C, Priority$_b$)*
3. If a conflict is detected (e.g., a permission and a prohibition), the system identifies, extracts and displays the rules responsible for this situation;
4. For an (*s*, *a*, *o*), we consider (among all the *is_permited* (*s*, *a*, *o*, <u>*P*</u>)), the one that has the greatest priority. We do the same for all the *is_prohibited*(*s*, *a*, *o*, <u>*P*</u>), *is_recommanded* (*s*,*a*, *o*, <u>*P*</u>) and *is, _obliged*(*s*, *a*, *o*, <u>*P*</u>). Then, among these four predicates (decisions), we recuperate the one that has the greatest priority. If there is equality, we consider that *is_prohibited* is more critical than *is_obliged,* more critical than *is_recommanded,* more critical *than is_permited.*
5. The system suggests relaxations (corrections, alternatives) by inviting the user to accept the decision calculated in the previous step, or to reformulate or to change the priority of one or several rules involved in the conflict.
6. The system takes into account these corrections, re-checks the coherence of the security policy and –if the

problem is solved– saves the new rule in the base that contain the security rules.

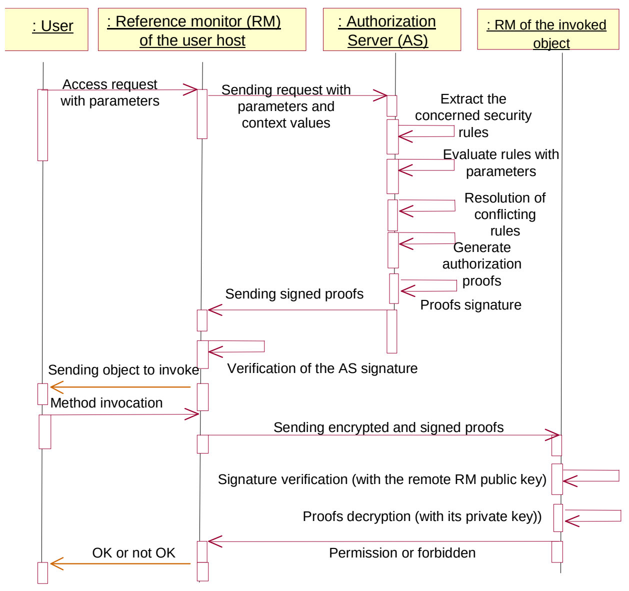
Fig. 11 gives examples of rules implemented with the Prolog language).

```
/* Examples of rules describing the system*/
org(hospital).
user(jean).
/* Examples of general relationships */
subject(X):-org(X).
subject(X):-person(X)
/* Examples of rules Multi-OrBAC relationships*/
Play(David, Physician-in-OrgA).
Consulting Physician(Jean,David).
age(Jean,2).
/* Examples of inheritance rules */
Play_inheritance(A,X-in-Z) :- sub_role(Y,Z),Play(A,X-in-Y)
Play_inheritance(A,X-in-Y) :- sub_org(B,A),Play(B,X,Y).
/* Adding an entity or a predicate */
Add(org(A)) :- \+org(A),asserta(org(A)),!.
Add(org(A)) :-
org(A),nl,write('No:'),nl,write('org('),write(A),write(')'),tab(1),write('a
lready'),tab(1),write('exist.').
```

Fig. 11: Examples from our implementation.

VI. A CASE STUDY

In the last sections, we have identified the static, conceptual as well as the logical aspects of the Multi-OrBAC model. In this section, we give the dynamic and architectural views. In particular, we explain the authorization's steps and we derive a secure architecture based on Multi-OrBAC security policy. Fig. 12 presents the scenario that we have implemented using the Java language.



: User
: Reference monitor (RM) of the user host
: Authorization Server (AS)
: RM of the invoked object
Access request with parameters
Sending request with parameters and context values
Extract the concerned security rules
Evaluate rules with parameters
Resolution of conflicting rules
Generate authorization proofs
Proofs signature
Sending signed proofs
Verification of the AS signature
Sending object to invoke
Method invocation
Sending encrypted and signed proofs
Signature verification (with the remote RM public key)
Proofs decryption (with its private key))
Permission or forbidden
OK or not OK

Fig. 12: the authorization phase description.

In this UML sequence diagram, we identify two basic components:

- The *authorization server* (AS): consults the security policy, extracts the relevant security rules, evaluates these rules with the current access parameters, invokes the conflict resolution process (cf. section 5.4), and generates the corresponding credentials.
- The *reference monitor* (RM, initially introduced in the Orange Book [27]): checks the authentication parameters and verifies whether each access request is accompanied by a valid authorization proof (e.g., a credential delivered by the authorization server).

When starting the application, the user chooses the R*i*O that he wants to play (e.g., radiologist in Org_A). Once authenticated (by the RM), he sends to the AS a request that contains parameters such as his R*i*O and the organization he wants to communicate with (suppose that this organization is Org_{dest}). The AS interrogates the database that contains the security rules. Actually, the table "policy" has the following scheme *"Access_Modality; RiO; AiO; View; Context"*. A security rule could thus be (*Permission*, Physician-*in-*Org_A, Reading-in-Org_B, MedicalRecord, disaster). Afterwards, the AS:

- Generates a ticket; the latter is a collection of permissions, e.g., {$\boldsymbol{R_a iO_b}$; (*Permission*, $A_\alpha iOrg_{dest}$, $view_c$, $context_1$); (*Permission*, $A_\beta iOrg_{dest}$,, $view_d$, $context_2$)}; this ticket is deduced from the two rules {*Permission*, ($\boldsymbol{R_a iO_b}$, $A_\alpha iOrg_{dest}$, $view_c$, $context_1$); *Permission*, ($\boldsymbol{R_a iO_b}$, $A_\beta iOrg_{dest}$,, $view_d$, $context_2$)}.
- Encrypts it with the public key of the RM of Org_{dest} (so that only the allowed addressee "i.e., the organization possessing the object" can decrypt the message with his private key), and
- Signs it with its private key (to prove that the ticket has been delivered by the AS).

When the user receives the ticket, he sends it to Org_{dest}. The RM of Org_{dest} verifies the AS signature, decrypts the ticket, and allows or forbids access. If the user wants to access objects belonging to other organizations, he asks the AS for other tickets.

Fig. 13 describes the global architecture while Fig. 14 presents the UML deployment diagram. The latter gives more details about the physical components: files, databases, interfaces, etc.

Basically, we have used a simplified architecture with two hospitals Org_1 and Org_2 (Fig. 13).

Org_1 uses a database system while Org_2 uses XML files. The global security policy is contained in an XML database and consulted by an AS. Remember that this policy contains Org_1 specific rules, Org_2 specific rules and coalition rules that ware negotiated between Org_1 and Org_2. The coalition rules could essentially concern the resources to share, e.g., emergency data (of Org_1's patients) that could be consulted by Org_2's clinicians in case of disaster, strike, etc.

The next section compares Multi-OrBAC with other important works. We first take an interest in Role-Based Trust Management (RBTM), and then we implement Multi-OrBAC using XACML.

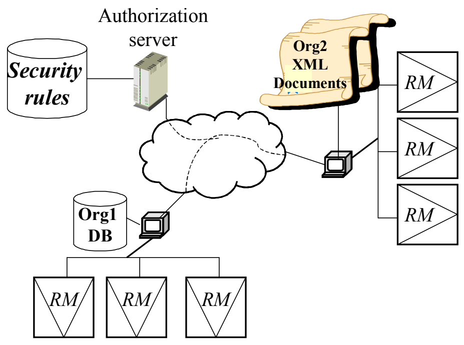


Fig. 13: System architecture.

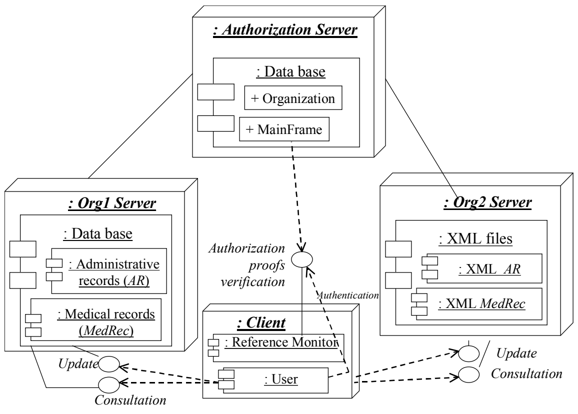


Fig. 14: The deployment diagram.

VII. DISCUSSION

A. Multi-ORBAC vs RBTM

Even if our implementation concerns a simplified scenario, we can imagine other applications where Multi-ORBAC could be useful. Globally, Multi-OrBAC is adapted to multicentric collaboration systems. In these systems users belonging to an organization need to access resources controlled by other organizations. For example, organizations participating to a particular project could grant access to their resources to users belonging to other organizations participating in the project, according to their roles.

Works on RBTM (Role-Based Trust Management, [2] has modelled this type of scenario by using role delegations and role-mapping across multiple collaborating organizations. Typically, each organization can delegate local roles to users belonging to other organizations. However, neither the role-mapping nor the delegation process is intuitive in heterogeneous and dynamic systems. Moreover, RBTM only abstract subject (by roles), while Multi-OrBAC expresses the whole security policy with abstract entities only. In this way, unlike RBTM, Multi-OrBAC: completely separates the representation of the security policy from its implementation; provides a global and homogeneous view of the security policy; improves the management of the security policy and reduce considerably its complexity.

B. Multi-ORBAC vs XACML

Multi-OrBAC could be integrated perfectly into a XACML architecture (Fig. 15) [28, 29], as the RM (Fig. 12) is quite similar to the Policy Enforcement Point (PEP), and the AS plays the roles of Policy Decision Point (PDP) and Policy Access Point (PAP). In this architecture, an access request arrives at the PEP with the parameters (R*i*O, AiOrg_{dest}, V, C). The PEP creates an XACML request and sends it to the Policy Decision Point (PDP), which evaluates the request and sends back a response. The response can be either access permitted or denied, with the appropriate obligations or recommendations.

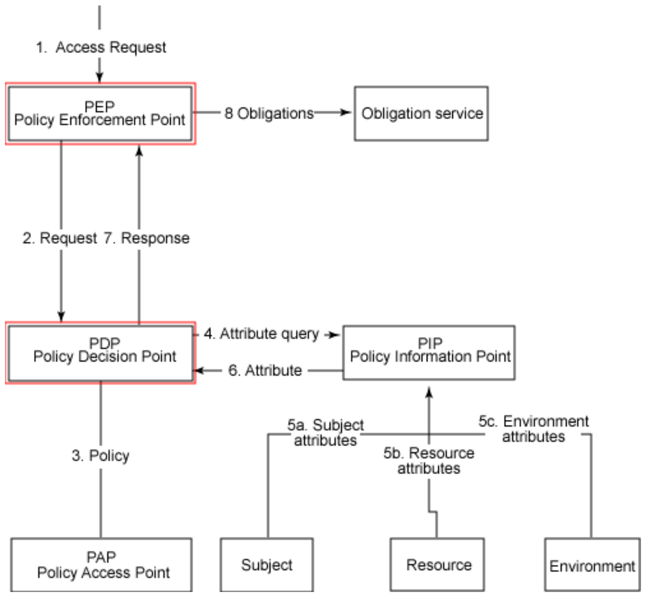


Fig. 15: The XACML Architecture.

The PDP arrives at a decision after evaluating the relevant policies (i.e., the policies concerning R*i*O, AiOrg_{dest}, ViOrg_{dest}) and the rules within them.

To get the policies, the PDP uses the PAP to extract the security rules (e.g., *Permission*(R*i*O, V*i*O, A, C)). The PDP may also invoke the Policy Information Point (PIP) service to retrieve the attribute values related to the subject, the resource, or the environment (the context). This corresponds to evaluate the associations *Belong_to* (α,AiO_2), *Correspond_to*(o, ViO_2), *Is_true*(CiO_2), etc.

The authorization decision arrived at by the PDP is sent to the PEP. The PEP:

- fulfils the obligations and/or inform the subject about the recommendations, and,
- based on the authorization decision sent by PDP, either permits or denies access.

Fig. 16 describes the components of a Multi-OrBAC implementation based on XACML. The differences from a traditional XACML implementation are:

- In some cases, the PDP can associate recommendations to its decision.

- The conditions have the form *Play (s, RiO_1)* ∧ *Correspond_to(o, ViO_2)* ∧ *Belong_to (α,AiO_2)* ∧ *Is_true(CiO_2)*;
- The rules combining algorithms combine the effects of all the rules in a policy to arrive at a final authorization decision. XACML defines the following algorithms: deny-overrides, permit-overrides, ordered-permit-overrides and first-applicable. In our implementation, we add the algorithms defined in Sections 5.3 and 5.4.
- A target[1] contains the instances of the triplet (*RiO*, *AiO*, *ViO*).

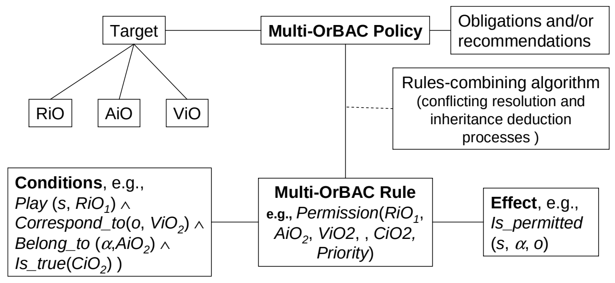

Target
Multi-OrBAC Policy
Obligations and/or recommendations
Rules-combining algorithm (conflicting resolution and inheritance deduction processes)
RiO
AiO
ViO
Conditions, e.g., Play (s, RiO1) ∧ Correspond_to(o, ViO2) ∧ Belong_to (α,AiO2) ∧ Is_true(CiO2))
Multi-OrBAC Rule e.g., Permission(RiO1, AiO2, ViO2, , CiO2, Priority)
Effect, e.g., Is_permitted (s, α, o)

Fig. 16: The components of a Multi-OrBAC implementation based on XACML.

VIII. CONCLUSIONS AND PERSPECTIVES

Multi-OrBAC is an extension of OrBAC that enables a better access control for collaborative organizations in distributed and heterogeneous contexts. In Multi-OrBAC, security rules are specified only through abstract entities. It can thus improves the management of the security policy and reduce considerably its complexity. Moreover, Multi-OrBAC is well-adapted to multicentric collaboration systems where external users access to shared resources.

A logical model (based on a Logical Programming by Constraints) was associated to *Multi-OrBAC*. The latter is used to specify the policy, derive permissions, detect and solve conflicting situations.

Besides, the Multi-OrBAC UML modelling helps to overcome the lack of expressiveness of OrBAC: Multi-OrBAC takes advantage of object-oriented facilities such as aggregation, inheritance, composition, etc. Moreover, the UML modelling of the authorization and deployment steps was very useful for deriving a Multi-OrBAC driven architecture.

We have used Java to implement the interfaces as well as the authentication and authorization steps, while the administration and conflicting detection (and resolution) steps were delegated to an engine based on the Prolog language.

Finally, we have shown how Multi-OrBAC could be incorporated in a XACML implementation.

Now, we are looking for other issues such as:

- Implementing a Multi-OrBAC security policy in a real large-scale system and calculating its complexity;
- automatically converting the UML conception into a logical (i.e., formal) one;
- defining an administration model associated to Multi-OrBAC; and finally,
- studying the federation process. Fig. 17 addresses a global view of the most relevant steps to perform.

[1] A target helps in determining whether the policy is to be evaluated for the current access request.

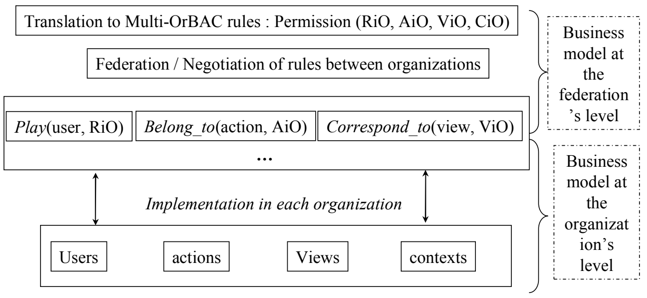

Translation to Multi-OrBAC rules : Permission (RiO, AiO, ViO, CiO)
Federation / Negotiation of rules between organizations
Play(user, RiO)
Belong_to(action, AiO)
Correspond_to(view, ViO)
...
Implementation in each organization
Users
actions
Views
contexts
Business model at the federation 's level
Business model at the organization's level

Fig. 17: The federation scheme.

AKNOWLEDGMENT

This work is partially supported by the French SATIN ACI and by the European project CRUTIAL.

REFERENCES

[1] *Common Criteria for Information Technology Security Evaluation, v3, Part 1: Introduction and general model*, 79 p., ISO/IEC 15408-1, July 2005.

[2] A. Abou El Kalam, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, C. Saurel, "OrBAC", *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, Como, Italy, 4-6 June 20, IEEE Comp Society Press, pp. 120-131.

[3] N. Li, J.C. Mitchell, W.H. Winsborough, Design of A Role-based Trust-management Framework, *IEEE Symposium on Security and Privacy*, pp. 114-130. IEEE Computer Society Press, May 2002.

[4] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, ISBN 0-201-57168-4, USA, 1999.

[5] D.E. Bell, L.J. LaPadula, *Secure Computer Systems*, MTR 2997, MITRE corp., USA, 1976.

[6] M.A. Harrison, W.L. Ruzzo and J.D. Ullman, "Protection in Operating Systems", *Communication of the ACM,* 19(8), pp. 461-471, august 1976.

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

[8] D. Ferraiolo, R. Sandhu, S. Gavrila, D.Kuhn, R. Chandramouli "A Proposed Standard for Role-Based Access Control", *ACM Transactions on Information and System Security,* v 4, n° 3, 2001.

[9] R.S. Sandhu, "Role Hierarchies and Constraints for Lattice-Bases Access Controls", in *4th European Symposium on Research in Computer Security*, Rome,

Italy, September 25-27, pp. 65-79, ISBN 3-540-61770-1, Springer-Verlag, 1996.

[10] G. Ahn and R. Sandhu, "Role-Based Authorization Constraints Specification", *ACM Transactions on Information and System Security,* vol. 3, n° 4, November 2000, pp. 207-226.

[11] J. Vitek, C. Jensen A View-Based Access Control Model for CORBA, *Secure Internet Programming*, LNCS 1603, Springer 1999.

[12] R. Thomas and R. Sandhu. Task-based Authorization Controls (TBAC), *11th IFIP Working Conference on Database Security*, Lake Tahoe, California, USA, 1997.

[13] B.N Schilit, M. Theimer, Disseminating Active Map Information to Mobile Hosts. *IEEE Network*, 8(5), 94, p. 22–32.

[14] M. Covington, W. Long, S. Srinivasan, A.K. Dey, M. Ahamad, G.D. Abowd, "Securing Context-Aware Applications Using Environment Roles", *ACM Symposium on Access Control Models and Technologies,* Chantilly, USA. May 3-4, 2001.

[15] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, "A Temporal access control mechanism for database systems", *IEEE Transactions on Knowledge and Data Engineering*, V8, 1996.

[16] J.B. Warmer and A. Kleppe, *The Object Constraint Language*, Addison-Wesley, ISBN 321179366, USA, 2003.

[17] Jaffar and J.L. Lassez, "*Constraint Logic Programming*", Communication of the ACM, vol. 33, N° 7, pp. 52 – 68, July1990.

[18] P.V. Hentrayck, "Constraint Satisfaction in Logic Programming", MIT Press 1989.

[19] R. Kowalski, "*Predicate Logic as Programming Language*", in Proc. IFIP Congress, 569-574 pp., North Holland, Amsterdam, 1974.

[20] A. Colmerauer, "*An introduction to Prolog III*", Comm. of the ACM, v. 33, N° 7 pp. 70-90, 90.

[21] rfc3198, *Terminology for Policy-Based Management*, November 2001, available at: http://www.faqs.org/rfcs/rfc3198.html

[22] J. D. Moffett and M. S. Sloman, "Policy Conflict Analysis in Distributed System Management," *Organizational Computing*, 1993.

[23] E. C. Lupu and M. Sloman, "*Conflicts in policy-based distributed systems management*" Soft. Eng., vol. 25, 1999.

[24] M. Strembeck, "Conflict Checking of Separation of Duty Constraints in RBAC" *Conference on Software Engineering*, 17-19 february 2004, Innsbruck, Austria.

[25] E. Bertino, S. Jajodia et P. Samarati, "Supporting Multiple Access Control Policies in Database Systems" *IEEE Symp. on Security and Privacy*, Oakland, 1996.

[26] Y. Deswarte, N. Abghour, V. Nicomette, D.Powell, "An Intrusion-Tolerant Authorization Scheme for Internet Applications", Workshop on Intrusion Tolerant System*s*, Washington (USA), 23-26 june 2002, pp. C1.1-C1.6.

[27] TCSEC, *Trusted Computer System Evaluation Criteria*, 122 pp., Department of Defense (DoD), DoD Standard, DoD 5200.28-STD, 1985.

[28] A; Matheus. "How to Declare Access Control Policies for XML Structured Information Objects using OASIS" *hicss*, v. 7, no. 7, pp. 168a, 2005.

[29] OASIS, *XACML Specification V*1.1, OASIS: www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf, 24 July 2003.