

Modelling contextual security policies

Frédéric Cuppens, Nora Cuppens-Boulahia
GET/ENST Bretagne, 2 rue de la Châtaigneraie,
35512 Cesson Sévigné Cedex, France
{firstname.lastname}@enst-bretagne.fr

Abstract

As computer infrastructures become more complex, security models must provide means to handle more flexible and dynamic requirements. In the Organization Based Access Control (OrBAC) model, it is possible to express such requirements using the notion of context. In OrBAC, each security rule (permission, prohibition, obligation or dispensation) only applies in a given context. A context is viewed as an extra condition that must be satisfied to activate a given security rule. In this paper, we present a taxonomy of different types of context and investigate the data the information system must manage in order to deal with these different contexts. We then explain how to model and evaluate them in the OrBAC model.

1. Introduction

Current computer infrastructure are becoming more and more complex and difficult to manage securely. It does no longer correspond to monolithic architecture but instead must manage a set of virtual communities that want to inter-operate and share resources. A virtual community is a composition of heterogeneous and independently designed and managed sub-organizations. When setting up a new community, it is necessary to identify participant roles, so Role Based Access Control (RBAC) model [40] provides concepts that are useful in this area.

However, security policies must also be adapted to deal with new requirements: security rules in these policies are no longer static but dynamic, depending on the context. They must be also self-adaptive with respect to temporal conditions, the user's location, previous behavior of this user, etc. Several different organizations must be able to express their own policies and the security policy model must be able to manage these different policies within a single framework. Classical access control models [7, 28, 40] are not sufficiently flexible to specify such context-dependent requirements.

In this paper, we shall show how the Organization Based Access Control (OrBAC) model [34] is useful to deal with some of these new requirements. In OrBAC, the security policy does not directly apply to subject, action and object. Instead, it defines permissions that applies within an *organization* to control the *activities* performed by *roles* on *views*. For instance, the policy might specify that role *physician* has permission to perform activity *consult* on view *medical record*.

In OrBAC, specification of a security policy is actually not restricted to permissions. A security policy may include four different types of security rule: permission, prohibition, obligation and dispensation. Intuitively, a prohibition is a negative permission implying that one must not perform some action. An obligation is associated with an action some-

one must perform and is usually triggered when some conditions are satisfied. Finally, a dispensation is a negative obligation implying that one may not perform some action.

The OrBAC model also allows the administrators to specify complex security rules since one can consider that each security rule only applies in some given *contexts*. Our objective is to further investigate this notion of context. To activate a given security rule, the subject, the action and the object must separately satisfy some conditions. In the OrBAC model, these conditions are that the subject must be assigned to a given role, the object must be used in a given view and the action implements some given activity. Besides these conditions, there are extra conditions that must be satisfied to activate a security rule. These extra conditions may be related to very different notions, such as temporal or spatial requirements. We call *context* such extra conditions.

In the following, we first investigate what kind of information a given information system must manage to provide means to deal with contextual conditions. Based on this analysis, we present several types of context – temporal, spatial, prerequisite, provisional and user-declared contexts – and explain how to model them in the OrBAC model. Even if there has been a lot of work dealing with various kinds of context (see section 5 for a comparison and discussion), this is the first time that such different contexts may be expressed within a unique access control model. We then address the problem of context evaluation. Generally, security models are based on Datalog rules [45] which guarantee the decidability of query evaluation and its termination in polynomial time. Using Datalog, queries are pre-computed iteratively using a bottom-up strategy until convergence to a fixed-point. However, as suggested in [6], it would not be possible to express most contextual conditions, for instance temporal or spatial contexts, using Datalog rules. This is why we suggest an hybrid strategy that combines the bottom up approach of Datalog with the top-down strategy as defined in the SLG algorithm [44] to evaluate contextual conditions. This hybrid strategy that also guarantees decidability in polynomial time is further explained in section 4.

The remainder of this paper is organized as following. Section 2 recalls the OrBAC model. Section 3 presents a taxonomy of different types of context and how to model them in the OrBAC model. Section 4 discusses decidability in OrBAC and presents an efficient strategy for query evaluation that combines bottom-up and top-down algorithms. Section 5 provides a comparison with related work. Finally, section 6 concludes the paper.

A preliminary version of this paper appeared in [19].

2. OrBAC model

2.1 Preamble

The final objective of a security policy is to specify the security rules (*permissions*, *prohibitions*, *obligations* or *dispensations*) that control the *actions* performed by *subjects* on *objects*. Using a logical notation, this might be represented by a set of facts having the following forms:

- $Is_permitted(s, \alpha, o)$,
- $Is_prohibited(s, \alpha, o)$,

- $Is_obliged(s, \alpha, o)$ or
- $Is_dispensed(s, \alpha, o)$.

meaning that a given subject s is permitted (resp. prohibited, obliged or dispensed) to perform a given action α on a given object o . For instance, the fact $Is_permitted(John, read, Paul_medical_record)$ specifies that John is permitted to read Paul's medical record.

However, enumerating all these facts is a quite fastidious and difficult to manage task. In particular, each time a new subject, or a new object, or a new action is created, it is necessary to explicitly insert new facts specifying the security rules associated with this new subject, object or action.

To simplify management of a security policy, rule based languages have been proposed [31, 20, 26]. These languages are based on first order logic with various restrictions to guarantee the decidability of logical derivation in these languages. For instance, the rules must respect syntactic restrictions of Datalog [45]. Using a rule based language, an access control policy is represented by a set of rules having the following forms:

- $\forall s, \forall \alpha, \forall o, (Condition \rightarrow Is_permitted(s, \alpha, o))$
- $\forall s, \forall \alpha, \forall o, (Condition \rightarrow Is_prohibited(s, \alpha, o))$
- $\forall s, \forall \alpha, \forall o, (Condition \rightarrow Is_obliged(s, \alpha, o))$
- $\forall s, \forall \alpha, \forall o, (Condition \rightarrow Is_dispensed(s, \alpha, o))$

meaning that, for every subject s , action α and object o , if a given condition is satisfied*, then subject s is permitted (resp. prohibited, obliged or dispensed) to perform action α on object o .

Let us now further analyze the structure of $Condition$ in the above rules. We suggest structuring $Condition$ as following:

$$cond_subject(s) \wedge cond_action(\alpha) \wedge cond_object(o) \wedge constraint(s, \alpha, o)$$

where:

- $cond_subject(s)$, $cond_action(\alpha)$ and $cond_object(o)$ are respectively the conditions the subject s , the action α and the object o must separately satisfy so that the corresponding rule applies.
- $constraint(s, \alpha, o)$ is an additional condition that joins subject s , action α and object o . Satisfying the constraint is necessary to activate the rule.

For instance, let us consider the rule “a physician is permitted to consult his or her patient's medical record”. In this case, $cond_subject(s)$, $cond_action(\alpha)$ and $cond_object(o)$ respectively correspond to the conditions that s is a physician, α is an action of consulting and o is a medical record. $constraint(s, \alpha, o)$ is a condition that joins subject s and object o (in this example, action α is absent from the constraint), namely o must be a record of physician s 's patient.

We shall now present OrBAC and explain how to model $cond_subject(s)$, $cond_action(\alpha)$, $cond_object(o)$ and $constraint(s, \alpha, o)$. For this purpose, we need first to present basic entities of the OrBAC model.

*Of course, this condition changes from one rule to another.

2.2 Basic concepts of OrBAC

The central entity in OrBAC is the entity *Organization*. Intuitively, an organization is any entity that is responsible for managing a security policy. Thus a company is an organization but concrete security components such as a firewall may be also viewed as an organization. An organization can also be seen as an organized group of subjects, assigned to some roles in the organization. Notice that a group of subjects does not necessarily correspond to an organization. More precisely, the fact that each subject is assigned to a role in the organization corresponds to some agreement between the subjects to form an organization.

The objective of OrBAC is to specify the security policy at the *organizational* level, that is independently of the implementation of this policy. Thus, instead of modelling the policy by using the concrete and implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects play in the organization. The role of a subject is simply called a *role* as in the RBAC model. On the other hand, the role of an action is called an *activity* whereas the role of an object is called a *view*.

The concept of View is different from the concept of Class. A class is a taxonomical concept used to structure the object descriptions, i.e. a class groups objects that have similar characteristics. By contrast, a view is an organizational concept used to structure the policy specification, i.e. a view groups objects on which the same security rules apply.

Finally, since security rules do not apply statically but their activation may depend on contextual conditions, the concept of *context* is explicitly introduced in OrBAC. Examples of context may be *Night*, *Working-Hours* or *Urgency* (see section 3 for further details about the context definition).

Therefore, in OrBAC, there are eight basic sets of entities: *Org* (a set of organizations), *S* (a set of subjects), *A* (a set of actions), *O* (a set of objects), *R* (a set of roles), *A* (a set of activities), *V* (a set of views) and *C* (a set of contexts).

We assume that $Org \subseteq S$ (that is any organization is a subject) and that $S \subseteq O$ (that is any subject is an object).

Any entities in the OrBAC model may have some attributes. This is represented by application-dependent predicates that associate the entities with the value of these attributes. For instance, if *s* is a subject, then $Name(s, n)$ is true if the value *n* represents the name of *s*, $Address(s, a)$ is true if the value *a* represents the address of *s*, etc.

Besides these application-dependent predicates used to model the attributes of entities, there are several OrBAC built-in predicates. Both application-dependent and OrBAC built-in predicates are used to specify security policies in the OrBAC model. The objective of the following sub-section is to present the different OrBAC built-in predicates.

2.3 Modelling the organization components

In OrBAC, $cond_subject(s)$, $cond_object(o)$ and $cond_action(\alpha)$ respectively correspond to conditions specifying that, in a given organization, a subject is empowered in a role, an object is used into a view and an action implements an activity. This is represented by the following OrBAC built-in predicates:

- *Empower* is a predicate over domains $Org \times S \times \mathcal{R}$.

If *org* is an organization, *s* a subject and *r* a role, then $Empower(org, s, r)$ means that *org* empowers subject *s* in role *r*. Unlike the TMAC model [43] or the RBAC model [40] which consider binary relations between organizations and subjects or between subjects and roles, notice that our model consider a ternary relation between organizations, subjects and roles. This is useful to model situations where a given subject is assigned to several roles but in different organizations.

- *Use* is a predicate over domains $Org \times O \times \mathcal{V}$.

If *org* is an organization, *o* is an object and *v* is a view, then $Use(org, o, v)$ means that *org* uses object *o* in view *v*. This ternary relation is useful to characterize organizations that give different definitions to the same view. For instance, take the case of the view “medical record” defined in *H1* hospital as a set of Word documents and defined in *H2* hospital as a set of tuples in a relational database.

- *Consider* is a predicate over domains $Org \times A \times \mathcal{A}$.

If *org* is an organization, α is an action and *a* is an activity, then $Consider(org, \alpha, a)$ means that *org* considers that action α implements the activity *a*. Since *Consider* is a ternary relation, different organizations may decide that the same action comes under distinct activities or that different actions come under the same activity. For instance, activity “consulting” corresponds, in *H1* hospital, to an action “read” that can be ran on data files whereas it corresponds, in *H2* hospital, to action “select” that can be performed on relational databases.

2.4 Role, activity and view definition

Instead of enumerating facts corresponding to instances of predicate *Empower*, it is also possible to specify *role definitions* which correspond to logical conditions that, when satisfied, are used to derive that some subjects are automatically empowered in the role associated with the role definition. Role definition may be viewed as an extension of the Attribute-Based User-Role assignment suggested in [4]. A role definition corresponds to a logical rule that has the *Empower* predicate in the conclusion and respects the Datalog restrictions (see section 4). For instance, a bookshop *BS* may consider that a subject is empowered in role *Gold_customer* if this subject is empowered in role *Customer* and if he or she has been a customer for more than ten years. This is modelled by the following role definition:

- $\forall s \in S, \forall d \in O,$
 $Empower(BS, s, Gold_customer) \leftarrow$
 $(Empower(BS, s, Customer) \wedge Membership(BS, s, d) \wedge d \geq 10)$

where $Membership(BS, s, d)$ is an application dependent predicate meaning that subject *s* has been a customer of bookshop *BS* for *d* years.

Activity and view definitions are similarly used to automatically manage assignment of action to activity and object to view. We assume that activity and view definitions also respect the Datalog restrictions.

2.5 Context definition

In section 2.1, we introduce the condition $constraint(s, \alpha, o)$ to model extra conditions a subject, an action and an object must satisfy to activate an access control rule. In OrBAC, these extra conditions are modelled through the notion of *context*. Each context has a name and its definition depends on the organization. Notice that we use the term “context” in a broad sense since it actually corresponds to any constraint that joins a subject, an action and an object*. For instance, in the health care domain, the entity *Context* will cover circumstances such as “urgency”, “medical research”, “attending physician”, etc.

From a conceptual point of view, entities *Organization*, *Subject*, *Object*, *Action* and *Context* are linked together by the OrBAC built-in predicate *Hold*:

- *Hold* is a predicate over domains $Org \times S \times A \times O \times C$.
If *org* is an organization, *s* a subject, α an action, *o* an object and *c* a context, then $Hold(org, s, \alpha, o, c)$ means that within organization *org*, context *c* holds between subject *s*, action α and object *o*.

The conditions required for a given context to be linked, within a given organization, to subjects, objects and actions will be formally specified by logical rules called *context definition*. Notice that context definitions generally do not correspond to Datalog rules and thus cannot be evaluated using the standard Datalog bottom-up strategy (see section 4 for further explanations about this issue).

A trivial example is the context *Nominal* which is true in every circumstance. It is defined as follows:

- $\forall org, \forall s \in S, \forall \alpha \in A, \forall o \in O,$
 $Hold(org, s, \alpha, o, Nominal)$

As another example, the context *Attending_physician* may be defined in hospital *H1* as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
 $Hold(H1, s, \alpha, o, Attending_physician)$
 $\leftarrow (Name_patient(o, p) \wedge Patient(s, p))$

that is, in hospital *H1*, the context “attending physician” holds between subject *s*, action α and object *o* if *o* is a record corresponding to a patient *p* (represented by the application dependent predicate $Name_patient(o, p)$) and *p* is a patient of subject *s* (represented by the application dependent predicate $Patient(s, p)$).

Modelling various types of context will be further analyzed in section 3.

2.6 Context Algebra

We define a context algebra to build conjunctive, disjunctive and negative contexts from more elementary contexts. For this purpose, we introduce three OrBAC built-in functions $\&$, \oplus and $\bar{\cdot}$. If c_1 and c_2 are two contexts, then $\&(c_1, c_2)$ is a conjunctive contexts, $\oplus(c_1, c_2)$ is a disjunctive context and \bar{c}_1 is a negative context. We shall use the infix notations $c_1 \& c_2$ and $c_1 \oplus c_2$ in place of the prefix notations $\&(c_1, c_2)$ and $\oplus(c_1, c_2)$. These composed contexts are defined by the following rules:

*Constraints do not apply systematically to all the parameters. For instance, we may have constraints that simply join subject and object or even constraints that are simply related to subject.

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C},$
 $Hold(org, s, \alpha, o, c_1 \& c_2) \leftarrow$
 $(Hold(org, s, \alpha, o, c_1) \wedge Hold(org, s, \alpha, o, c_2))$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c_1 \in \mathcal{C}, \forall c_2 \in \mathcal{C},$
 $Hold(org, s, \alpha, o, c_1 \oplus c_2) \leftarrow$
 $(Hold(org, s, \alpha, o, c_1) \vee Hold(org, s, \alpha, o, c_2))$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall c \in \mathcal{C},$
 $Hold(org, s, \alpha, o, \bar{c}) \leftarrow \neg Hold(org, s, \alpha, o, c)$

Evaluation of these composed contexts is further explained in section 4.

2.7 Global Constraints

In OrBAC it is also possible to define global constraints. A global constraint is distinct from the definition of contexts presented in the previous sections. A context is associated with a security rule and defines conditions that must be satisfied to activate this security rule. By contrast, a global constraint is a condition associated with the security policy. If a global constraint is not satisfied, then this means that this security policy is not *consistent*.

To specify global constraint in the OrBAC model, we use the OrBAC built-in predicate *Error*:

- *Error* is a predicate over the null domain, i.e. a predicate of cardinality zero.

Using the *Error* predicate, a global constraint corresponds to a logical rule whose conclusion is *Error*. If it is actually possible to derive *Error* from some global constraint, then the security policy is not consistent.

Separation of duty between roles is an example of frequently used constraints. To model separation of duty in OrBAC, we first introduce the following OrBAC built-in predicate *Separated_role*:

- *Separated_role* is a predicate over domains $Org \times \mathcal{R} \times Org \times \mathcal{R}$.

If org_1 and org_2 are organizations and r_1 and r_2 are roles, then $Separated_role(org_1, r_1, org_2, r_2)$ means that role r_1 in organization org_1 is separated from role r_2 in organization org_2 .

Using this *Separated_role* predicate, separation of duty is then modelled as the following global constraint:

- $\forall s \in S, \forall org_1 \in Org, \forall org_2 \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R},$
 $Separated_role(org_1, r_1, org_2, r_2) \wedge$
 $Empower(org_1, s, r_1) \wedge$
 $Empower(org_2, s, r_2)$
 $\rightarrow Error$

This constraint says that *Error* is true if (1) role r_1 in organization org_1 is separated from role r_2 in organization org_2 and (2) there is a subject s empowered in role r_1 in organization org_1 and (3) this subject is also empowered in role r_2 in organization org_2 .

In the following, we assume that the security policy is consistent, i.e. it is not possible to derive the *Error* predicate from any global constraints. If the security policy is updated so that it is possible to derive *Error*, then this update is rejected.

2.8 Policy definition

Using the materials presented in the previous sections, the *Condition* expression introduced in section 2.1 corresponds in the OrBAC model to formulas having the following form:

- $Empower(Org, s, R) \wedge Use(Org, o, V) \wedge$
 $Consider(Org, \alpha, A) \wedge Hold(Org, s, \alpha, o, C)$

where s , o and α are variables corresponding respectively to a subject, an object and an action and Org , R , V , A and C are constants corresponding respectively to an organization, a role, a view, an activity and a concept. For instance, the rule “in hospital H , a physician is permitted to consult his or her patient’s medical record” may be represented by a rule having the following form:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
 $(Condition \rightarrow Is_permitted(s, \alpha, o))$
 where, in this case, *Condition* is the following formula:
 $Empower(H, s, Physician) \wedge Use(H, o, Medical_record) \wedge$
 $Consider(H, \alpha, Consult) \wedge Hold(H, s, \alpha, o, Attending_physician)$

However, this is not exactly the way an access control policy is specified in the OrBAC model. In OrBAC, we go one step further by considering that the access control policy does not directly apply to subject, action and object. Instead, the access control policy is specified using the OrBAC built-in predicates *Permission*, *Prohibition*, *Obligation* and *Dispensation* defined as follows:

- *Permission*, *Prohibition*, *Obligation* and *Dispensation* are predicates over domains $Org \times \mathcal{R} \times \mathcal{A} \times \mathcal{V} \times \mathcal{C}$
 If org is an organization, r a role, a an activity, v a view and c a context then $Permission(org, r, a, v, c)$ (resp. $Prohibition(org, r, a, v, c)$, $Obligation(org, r, a, v, c)$ or $Dispensation(org, r, a, v, c)$) means that in organization org role r is granted permission (resp. prohibition, obligation or dispensation) to perform activity a on view v within context c .

The predicates *Permission*, *Prohibition*, *Obligation* and *Dispensation* enable a given organization to specify permissions, obligations and prohibitions between roles, activities and views in a given context. Now, triples that are instances of the OrBAC built-in *Is_permitted* are logically derived from permissions granted to roles, views and activities by the relationship *Permission*. This is modelled by the following general rule called Concrete Permission Derivation*:

- $\forall org \in Org, \forall s \in S, \forall o \in O, \forall \alpha \in A, \forall r \in \mathcal{R}, \forall v \in \mathcal{V}, \forall a \in \mathcal{A}, \forall c \in \mathcal{C},$
 $Permission(org, r, a, v, c) \wedge$
 $Empower(org, s, r) \wedge Use(org, o, v) \wedge$
 $Consider(org, \alpha, a) \wedge$
 $Hold(org, s, \alpha, o, c)$
 $\rightarrow Is_permitted(s, \alpha, o)$

*Three similar general rules respectively called Concrete Prohibition, Obligation and Dispensation Derivation exist to derive instances of *Is_prohibited*, *Is_Obligated* and *Is_dispensed* from relationships *Prohibition*, *Obligation* and *Dispensation*.

that is s is permitted to perform α on o , if (1) organization org , within the context c , grants role r permission to perform activity a on view v and (2) org empowers subject s in role r and (3) org uses object o in view v and (4) org considers that action α implements activity a and (5) within org , the context c holds between s , α and o .

In OrBAC, we also assume that if a subject is obliged to perform a given action on an object, then this subject must be permitted to perform this action on this object. This principle is modelled by the following logical rule:

- $\forall s \in S, \forall o \in O, \forall \alpha \in A,$
 $Is_obliged(s, \alpha, o) \rightarrow Is_permitted(s, \alpha, o)$

Similarly, if a subject is prohibited to perform a given action on an object, then this subject must be dispensed from performing this action on this object:

- $\forall s \in S, \forall o \in O, \forall \alpha \in A,$
 $Is_prohibited(s, \alpha, o) \rightarrow Is_dispensed(s, \alpha, o)$

2.9 Conflict management

In OrBAC, it is thus possible to specify a security policy by combining permission, prohibition, obligation and dispensation. This provides a very expressive and flexible framework to express various security requirements. For instance, it is possible to specify general security rules corresponding to permissions (resp. obligations) and then specify possible exceptions to these general rules corresponding to prohibitions (resp. dispensations).

However, when a security policy model includes the possibility to specify both permissions, prohibitions, obligations and dispensations, some conflicts may occur. We identify the following possible conflicts:

- Conflict between permission and prohibition. This conflict occurs when it is possible to derive, for some subject s , action α and object o :
 $Is_permitted(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o)$
- Conflict between obligation and dispensation. This conflict occurs when it is possible to derive, for some subject s , action α and object o :
 $Is_obliged(s, \alpha, o) \wedge Is_dispensed(s, \alpha, o)$
- Conflict between obligation and prohibition. This conflict occurs when it is possible to derive, for some subject s , action α and object o :
 $Is_obliged(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o)$

However, since an obligation implies a permission, if there is a conflict between an obligation and a prohibition, then there is also a conflict between a permission and a prohibition.

So, a conflict between an obligation and prohibition is actually not a *primary* conflict.

Thus, we have two primary conflicts to manage in OrBAC: (1) conflict between permission and prohibition and (2) conflict between obligation and dispensation.

When some conflicts occur in the policy, we need strategies to solve these conflicts. The basic principle to define such strategies is assigning priorities to security rules. Thus, in case of conflict, the security rule with higher priority takes precedence over the other security rule. In our approach, we actually manage conflicts at the abstract level and provide sufficient conditions to guarantee that the absence of conflict at the abstract level also

guarantee the absence of conflict at the concrete level. We shall not further develop this issue here, but this problem is extensively investigated in [16].

2.10 Policy application

The purpose of a security policy is to control how subjects access to objects. In this section, we assume the existence of a trusted reference monitor which is in charge of enforcing the security policy. We also consider that when a subject wants to perform an action on an object, this subject must first send a request to this reference monitor.

The objective of this section is to define the security requirements to be enforced by this reference monitor. We first introduce the following built-in predicates *Request*, *Accept* and *Violation*:

- *Request*, *Accept*, *Deny* and *Violation* are predicates over domains $S \times A \times O$
If s is a subject, α an action and o an object, then $Request(s, \alpha, o)$ means that s is requesting to perform action α on object o , $Accept(s, \alpha, o)$ means that this request is accepted, $Deny(s, \alpha, o)$ means that this request is denied and $Violation(s, \alpha, o)$ means that the security policy is violated.

Using these predicates, we define the following security requirements:

- **Close policy requirement:** A request must be accepted only if it is permitted by the policy:
 $\forall s \in S, \forall o \in O, \forall \alpha \in A,$
 $Request(s, \alpha, o) \wedge Is_permitted(s, \alpha, o) \rightarrow Accept(s, \alpha, o)$
If it is not possible to explicitly derive $Accept(s, \alpha, o)$ then $Deny(s, \alpha, o)$ is implicitly derived.
- **Open policy requirement:** A request must be denied only if it is prohibited by the policy:
 $\forall s \in S, \forall o \in O, \forall \alpha \in A,$
 $Request(s, \alpha, o) \wedge Is_prohibited(s, \alpha, o) \rightarrow Deny(s, \alpha, o)$
If it is not possible to explicitly derive $Deny(s, \alpha, o)$ then $Accept(s, \alpha, o)$ is implicitly derived.
- **Obligation requirement:** The policy is violated when a subject does not request an access when it is obliged to do so:
 $\forall s \in S, \forall o \in O, \forall \alpha \in A,$
 $Is_obliged(s, \alpha, o) \wedge \neg Request(s, \alpha, o) \rightarrow Violation(s, \alpha, o)$

When implementing the reference monitor, only one of the *close* or *open* security requirement must be implemented. This is an administrative decision to choose between one of these two requirements.

Regarding the obligation requirement, it should be part of the security policy to specify what happens when a violation occurs. This generally corresponds to the activation of new security rules when such a violation occurs. Notice also that, in the above obligation requirement, we consider that a subject must *immediately* request an access when it is obliged to do so. In a more realistic setting, a subject may have some delay to request for an access before a violation occurs. See the Nomad model that shows how to specify and manage obligations associated with delays [18].

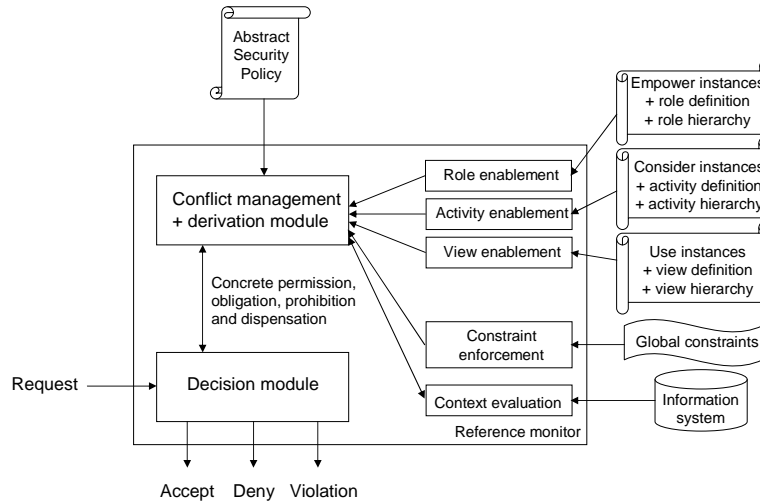


Figure 1: OrBAC global evaluation architecture

2.11 Summary

There are several ways to specify a security policy. A first way consists in explicitly enumerating a set of facts specifying the privileges (permissions, prohibitions, obligations and dispensations) of subjects to perform actions on objects. However, this approach is very difficult to manage. A second way is to specify a set of logical rules specifying the conditions that must be satisfied to derive that subjects are permitted, prohibited, obliged or dispensed to perform actions on objects. However, specifying this set of security rules is a complex and error prone task.

The OrBAC model suggests an hybrid approach summarized in figure 1. The abstract security policy is specified by enumerating a set of facts corresponding to privileges. However, these privileges do not concern subject, action and object but role, activity and view. Thus the reference module includes three components respectively called Role enablement, Activity enablement and View enablement that manage explicit instances (corresponding to facts) or implicit instances (derived from role, activity or view definitions) of predicates *Empower*, *Consider* and *Use*. As further explained in section 4, these three enablement components are based on the bottom-up evaluation strategy of Datalog. The reference monitor includes functionalities to enforce global constraints and manage conflicts. As mentioned in section 2.9, conflicts are managed at the abstract level so that we can guarantee that no conflict can occur at the concrete level. The reference monitor also includes a decision module that enforces the close (or open) policy requirement and the obligation requirement as defined in section 2.10. This decision module takes request as inputs and provides *accept*, *deny* or *violation* as outputs. To make its decision, the decision module requests the derivation module that applies the four general derivation rules shown in section 2.8 to derive what actions subjects are concretely permitted, prohibited, obliged or dispensed to perform on objects. When the decision depends on

some contextual conditions, the derivation module asks the context evaluation module for evaluation of the context associated with the abstract security rules. As explained in section 4, derivation and context evaluation modules are no longer based on the bottom-up evaluation strategy of Datalog but use a top-bottom strategy.

Notice that in [34], it is suggested to define hierarchies over roles but also organizations, activities and views, and to associate permission inheritance with these different hierarchies. This is modelled as follows:

- *Sub_role*, is a relation over domains $Org \times \mathcal{R} \times \mathcal{R}$.
If *org* is an organization, and r_1 and r_2 are roles, then $Sub_role(org, r_1, r_2)$ means that, in organization *org*, role r_1 is a sub-role (also called senior role) of role r_2 .
Permissions, prohibitions, obligations and dispensations are inherited through the role hierarchy. For instance, inheritance of permissions is modelled by the following rule:
 $\forall org \in Org, \forall r_1 \in \mathcal{R}, \forall r_2 \in \mathcal{R}, \forall v \in \mathcal{V}, \forall a \in \mathcal{A}, \forall c \in \mathcal{C},$
 $Permission(org, r_2, a, v, c) \wedge$
 $Sub_role(org, r_1, r_2)$
 $\rightarrow Permission(org, r_1, a, v, c)$
- *Sub_view* and *Sub_activity* are similarly defined as relations over domains $Org \times \mathcal{V} \times \mathcal{V}$ and $Org \times \mathcal{A} \times \mathcal{A}$.
- *Sub_organization* is a relation over domains $Org \times Org$.

How to manage inheritance of privileges through these different hierarchies is further explained in [17].

3. Context in OrBAC

3.1 Taxonomy of contexts

As we have just seen, we use contexts to express different types of extra conditions that control activation of rules specified in the security policy. In this section, we investigate the following contexts (see figure 2) and show how we model them in OrBAC:

- the *Temporal context* that depends on the time at which the subject is requesting for an access to the system,
- the *Spatial context* that depends on the subject location,
- the *User-declared context* that depends on the subject objective (or purpose),
- the *Prerequisite context* that depends on characteristics that join the subject, the action and the object.
- the *Provisional context* that depends on previous actions the subject has performed in the system.

As suggested in figure 1, we also assume that each organization manages some information system that stores and manages different types of information. To control context activation, this information system must provide the information required to check that conditions associated with the context definition are satisfied or not. The following list gives the kind of information related to the contexts we have just mentioned:

- A *global clock* to check the temporal context,

- the *subject environment* and the *software and hardware architecture* to check the spatial context,
- the *subject purpose* to check the user-declared context,
- the *system database* to check the prerequisite context,
- an *history* of the action carried out, to check the provisional context.

Figure 2 presents the correspondence between the contexts and the required data. If the information system does not provide some information in this list, then obviously the corresponding context cannot be managed by the access control policy.

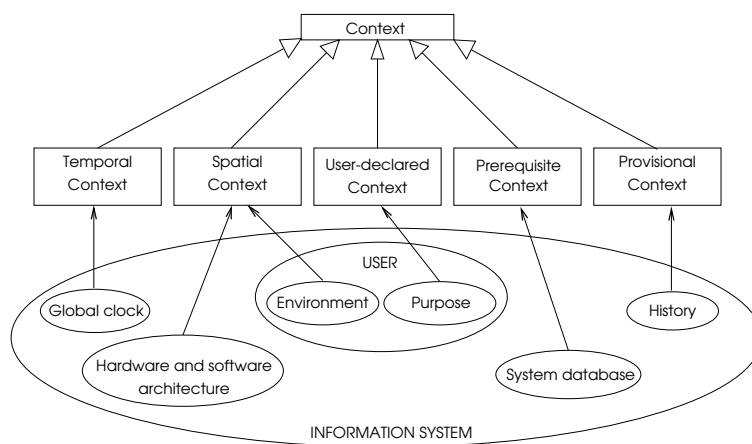


Figure 2: Context taxonomy and required data

Notice that in our approach, a context actually corresponds to a condition that joins a subject, an object and an action. However, one or several of these parameters may be optional. This means that we can, for instance, consider constraints that only apply to a subject, an object or an action. In this case, the constraint may be modelled by respectively defining “contextual” roles, activities or views. For instance, if we have a permission that applies to role physician in the temporal context “night”, we can consider a contextual role “night physician”. This approach has been suggested several times before in the literature (see [33] for example). However, in the general case, we argue that this approach leads to quite artificial roles (see section 5.2 for a more detailed discussion). Moreover, it does not applies when the constraint joins several parameters, for instance a subject and an object as for instance the *Attending-physician* context presented in section 2.5.

3.2 Temporal context

Principle

With temporal contexts, it should be possible to express that a given action made by a given user on a given object is authorized only at a given time, after or before a given time, or during a given time interval. The temporal conditions can correspond to a day of the week, or to a time of the day, etc. For instance, a physician within an hospital may be

allowed to access the medical record server only during the working hours, that is between 8:00AM and 19:00PM for example.

To validate a given query for an access, it is necessary to be able to evaluate the current time. Thus, we assume that the information system has a trusted clock, and that this clock can be queried at any time by the reference monitor to assess the temporal context of the query. We consider the entity clock as an object called *Global_clock*.

We associate the following attributes to *Global_clock*: *Time*, *Day*, *Week*, *Month*, *Year*. The corresponding predicates make it possible to obtain the current time, the current day, the current week, etc.

This is not the purpose of this section to define a complete set of predicates to be used to manage time but the interested reader can have a look at [38] for an extensive discussion.

Instead, our objective in this section is to show how various temporal contexts can be defined using these predicates. The general principle to build temporal contexts is to define functions that applies to the set T of times and return a temporal context. In the following sub-section, we give several examples to illustrate this principle.

However, notice that the semantics of a temporal context assessment may be subtle. For instance, let us consider a function *timeinterval*($t1, t2$). Usually, this function is evaluated using the system local clock. However, if the security policy needs to synchronize the timing across several specified points, an external timing source is necessary. Thus, as the semantics may differ from one environment to another with respect to the security requirements, to the access requestor and the requested organization, several supports to the temporal context assessment may be required (see [38] for further details).

Basic temporal contexts

We can first define two functions *before_time* and *after_time* that applies to the set T and return a temporal context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall t \in T, \forall t' \in T,$
 $Hold(org, s, \alpha, o, after_time(t)) \leftarrow (Time(Global_clock, t') \wedge t' \geq t)$
- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall t \in T, \forall t' \in T,$
 $Hold(org, s, \alpha, o, before_time(t)) \leftarrow (Time(Global_clock, t') \wedge t' \leq t)$

We can similarly define two functions *before_date* and *after_date* that apply to the set of *Date* and return a temporal context. We also consider a function *on_day* that applies to the set of *Day* and return a temporal context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall d \in Day,$
 $Hold(org, s, \alpha, o, on_day(d)) \leftarrow Day(Global_clock, d)$

Composed temporal context

Using the basic temporal contexts, we can define more complex temporal contexts using the context algebra, for instance:

- $night = after_time(23 : 00) \oplus before_time(8 : 00)$
- $weekend = on_day(saturday) \oplus on_day(sunday)$
- $working_hours = after_time(08 : 00) \& before_time(19 : 00) \& \overline{weekend}$

Notice that context definition actually depends on the organization. For instance, in an organization where employees works on Saturday but not on Monday, context *working_hours* would be defined as follows:

- $working_hours = after_time(08 : 00) \& before_time(19 : 00) \underline{\hspace{10em}}$
 $\& on_day(sunday) \& on_day(monday)$

Examples of permissions using temporal context

Let us consider the following rule: “In hospital *H1*, a physician is permitted to consult the medical record database *med_db* during working hours”. This is expressed by the following fact:

- $Permission(H1, physician, consult, med_db, working_hours)$

Let us now assume that the role *cardiologist* is also permitted to consult *med_db* on Sunday. This is expressed by the following fact:

- $Permission(H1, cardiologist, consult, med_db,$
 $working_hours \oplus on_day(sunday))$

If we assume that *cardiologist* is a senior role of *physician*, it would be actually sufficient to specify that *cardiologist* is permitted to consult *med_db* on Sunday, since the permission in temporal context *working_hours* will be inherited from *physician*.

3.3 Spatial context

Principle

Knowing the location from where the user makes the request can be useful to specify the access control policy. For example an hospital manager may be granted the right to read all employees’ payrolls. But she must read those payrolls in a secure area, for example in her own office. It thus reduces the possibility of curious employees being able to read their colleague’s payrolls over the manager’s shoulder. Spatial context is used to express this kind of condition.

We can distinguish two different types of spatial context. The physical spatial context and the logical spatial context. The first one corresponds to the physical location of the user, namely his or her office, a security area, a specific building, the country, etc. The logical spatial context corresponds to the “logical location” he or she stands in. For example, it can be the computer, the network or the sub-network, the cell in the case of radio communication such as in UMTS, etc.

In some cases, physical and logical spatial contexts are highly correlated. The network IP address from which a user is connected probably corresponds to a specific physical place such as a department area. Note that due to the expanding use of Global Position System (GPS) tools, it could be possible to locate a user or a terminal independently of the network.

If an organization allows its employees the use of Mobile IP, it is necessary to take into account from which network a request is emitted. A user will generally get less permissions if he or she is connected from a customer’s office. Moreover the development of wireless technologies such as Wi-Fi motivates this work. The security policy must make it possible to take into account the fact that a user is connected through a wire network, a wireless network, or on which Wi-Fi access point he or she is attached.

In the remainder of this section, we shall first present how to respectively specify physical and logical spatial contexts. We shall then give examples of how these contexts may be used when specifying a security policy.

Physical spatial contexts

As suggested by the Open GIS Consortium [12], we first assume the existence of a set SO of spatial objects (also called spatial features in [12]). Examples of spatial objects may be Rennes, France or the Office_232 in the hospital H1. Notice that a physical organization such as the hospital H1 can belong to the set SO .

We assume that spatial objects are associated with location functions so that it is possible to determine if a given subject is located in the area of a given spatial object. For this purpose, we introduce the OrBAC built-in predicate *Is_located*:

- *Is_located* is a predicate over domains $Org \times S \times SO$

If *org* is an organization, *s* is a subject and *so* is a spatial object, then *Is_located*(*org*, *s*, *so*) means that the location functions used in organizations *org* can determine that subject *s* is located in the area of spatial object *so*.

Notice that the truth of the *Is_located* predicate depends on the organization. This means that a given organization may be able to determine that some subject is located in the area of some spatial object which may not be the case for another organization.

We then define a function *location* that applies to the set SO of spatial objects and return a physical spatial context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall so \in SO,$
 $Hold(org, s, \alpha, o, location(so)) \leftarrow Is_located(org, s, so)$

This rule says that a given subject is performing a given action on a given object in the context *location*(*so*) if this subject is located in the area of spatial subject *so*.

In OrBAC, spatial objects as other objects are grouped into views. For instance, the fact *Use*(*H1*, *Office_232*, *Office*) means that the organization *H1* uses the spatial object *Office_232* in view *Office*. We say that a view *v* is a spatial view if it contains spatial objects and we denote SV the set of spatial views.

We can then generalize the function *location* so that it also applies to the set SV of spatial views and return a physical spatial context. This physical spatial context is defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall sv \in SV, \forall so \in SO,$
 $Hold(org, s, \alpha, o, location(sv)) \leftarrow (Use(org, so, sv) \wedge Is_located(org, s, so))$

This rule says that a given subject is performing a given action on a given object in the context *location*(*sv*) (where *sv* is a spatial view for instance *Office*) if this subject is located in the area of some spatial subject *so* and *so* is used in the spatial view *sv*.

It is also possible to define more “customized” contexts, for instance, the context *personal_office* may be defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall po \in SO,$
 $Hold(org, s, \alpha, o, personal_office) \leftarrow$
 $(Personal_office(org, s, po) \wedge Is_located(org, s, po))$

In this rule, we consider that *Personal_office*(*org*, *s*, *po*) is an application dependent predicate that says that, in organization *org*, the spatial object *po* is the personal office of subject *s*. So the rule says that a given subject is performing an action on an object in the context *personal_office* if this subject is located in his or her personal office.

Notice that the evaluation of context *personal_office* requires a query to the system

database to evaluate the predicate $Personal_office(org, s, po)$. So this context is actually an hybrid context, that is a combination of a physical spatial context and a prerequisite context (see section 3.5 below).

Logical spatial contexts

The logical spatial contexts depend on the characteristics of the network and computer infrastructure of the organization. The logical location of a subject depends on some of these characteristics such as on which host this subject is logged, to which subnetwork this host is connected, how this host is connected to the subnetwork, etc. We can also define the logical location of the object to be accessed by determining on which host this object is stored.

As suggested for instance in [46], it is possible to define a set of predicates to model the network and computer characteristics. For this purpose, we need:

- Predicates to determine the logical location of the subject, for example $Login(s, h)$ (subject s is logged on host h),
- Predicates to determine the logical location of the object, for example $Storage(o, h)$ (object o is stored on host h),
- Predicates to model the network infrastructure, for example $Location_zone(h, net)$ (host h is connected to the network net), $Host_os(h, os)$ (host h is running operating system os),
- Predicates to model the security of communication channels, for example $Authenticated(ch, ac)$ (authentication method ac has been successfully used during the initialization of channel ch), $Encryption(ch, en)$ (encryption method en is used for channel ch),
- Predicates to model application security, for example $Application(org, \alpha, t)$ (organization org has assigned trust level t to the action α).

To illustrate how these predicates may be used, let us consider a company C that has a secured area SA in which specific security requirements are enforced. For example, there is no possibility of optical eavesdropping [36]. SA corresponds to the name of a specific subnetwork. Users are allowed to consult certain documents on their laptop only in this area. If a given subnetwork address is allocated in this area, then the IP address of the terminal that is making a request is enough to locate it. Thus, using the above defined $Login$ and $Location_zone$ predicates, the logical context $in_secured_area$ can be defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall host \in O,$
 $Hold(C, s, \alpha, o, in_secured_area) \leftarrow$
 $Login(s, host) \wedge Location_zone(host, SA)$

A similar idea can be used in the case of Mobile IP. In this case, the local agent must manage the network where the mobile hosts are.

Let us consider another example. In a wireless network, some user is allowed to access a specific resource from everywhere but only with his or her own laptop. The attribute boolean $host_MAC$ is allocated to the entity user that indicates if the MAC address of the packet received is really the MAC address of the user's laptop. The context $on_personal_laptop$ is then defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O,$
 $Hold(C, s, a, o, on_personal_laptop) \leftarrow host_MAC(s)$

Notice that specific security mechanisms must be implemented to prevent a malicious user to bypass access control requirements by forging his or her own packets, and choosing the appropriate IP address or MAC address. This is not our objective to discuss such security issues in this paper.

However, these two examples only correspond to specific logical spatial contexts defined by some given organization. Using the above predicates and as suggested in [46], it would be more interesting to define generic contexts that specify a trust level associated with the execution of some action by a given subject on a given role. Thus, if TL is a set of trust level then, we could define a set of contexts $trust_level(l)$ for each $l \in TL$.

Defining these kind of contexts would be very useful, especially when several systems having different security policies have to inter-operate. To be effective, this approach must be based on an ontology to formally define the set of predicates to be used and a risk analysis to fix the definition of the $trust_level$ contexts. This issue clearly requires further work that goes beyond the scope of this paper.

Example of permissions using spatial contexts

Let us first consider the following rule: “In hospital $H1$, a physician located inside $H1$ is permitted to consult the medical record database med_db ”. This is expressed by the following fact:

- $Permission(H1, physician, consult, med_db, location(H1))$

Notice here that the organization $H1$ is used as a spatial object to define a physical spatial context.

Let us then assume that “a physician located outside $H1$ is permitted to consult the medical record of med_db but only on his or her personal laptop”. This is modelled as follows:

- $Permission(H1, physician, consult, med_db,$
 $\overline{location(H1)} \& on_personal_laptop)$

Here we have used a conjunctive context between a (negative) physical spatial context and a logical spatial context.

As a third rule “a hospital manager is permitted to consult payroll if he or she is located in his or her office or in a secure area”:

- $Permission(H1, manager, consult, payroll,$
 $in_personal_office \oplus in_secure_area)$

Finally, “a nurse is permitted to consult the prescription if he our she is located in a hospital service”:

- $Permission(H1, nurse, consult, prescription, location(hosp_service))$

In this last exemple, we used a spatial view $hosp_service$ to define a physical spatial context. For example, if $service_12$ is a service of hospital $H1$, then a nurse will be permitted to consult the nurse record when located in $service_12$.

3.4 User-declared context

Principle

In some circumstances, a subject according to the role he or she is empowered in the organization may be allowed to declare that he or she performs some activities in a given context. When declaring a context, a subject will obtain some specific permissions and possibly also some obligations or prohibitions. For instance, a subject empowered in the role medical researcher may be permitted to declare that he or she is performing an epidemiological analysis. By doing so, this subject will get the permission to have an access to some statistical database.

In our approach, user declared contexts are modelled as follows. Declaring a context actually corresponds to creating a special object called purpose object and inserting this object in a view called *Purpose* (if this insertion is permitted by the policy). Thus we first consider a set of purpose objects denoted PO . Purpose objects are used to describe the user-declared context activated by some subject. Purpose objects have two attributes *Recipient* and *Declared_purpose* defined as follows.

- *Recipient* is a predicate over domains $PO \times S$
If po is a purpose object belonging to PO and s is a subject, then $Recipient(po, s)$ means that s is the subject who takes advantage of the declared purpose po .
- *Declared_purpose* is a predicate over domains $PO \times PV$
Here, we assume the existence of a set of purpose values denoted PV . Examples of purpose values are *Medical_research*, *Epidemiology*, *Hemophilia*, *Cancer*, etc.
Thus, if po is a purpose object and pv is a purpose value then $Declared_purpose(po, pv)$ means that pv is the purpose value associated with the declared purpose po .

We then define a function *user_declared* that applies to the set PV of purpose values and return a user declared context defined as follows:

- $\forall org \in Org, \forall s \in S, \forall \alpha \in A, \forall o \in O, \forall po \in PO, \forall pv \in PV,$
 $Hold(org, s, \alpha, o, user_declared(pv)) \leftarrow$
 $Use(org, po, Purpose) \wedge$
 $Recipient(po, s) \wedge Declared_purpose(po, pv)$

that is, in organization org , subject s performs action α on object o in the user declared context $user_declared(pv)$ if there is a purpose object po used in view *Purpose* by organization org such that s is the recipient associated with po and pv is the declared purpose associated with po .

Notice that it is possible to consider sub-views of view *Purpose*. In particular, we consider that each purpose value pv in PV is also a sub-view of *Purpose* that corresponds to restriction of the view *Purpose* on the declared purpose value pv^* . This is defined by the following view definition:

- $\forall org \in Org, \forall po \in PO, \forall pv \in PV,$

*This corresponds to a classical abuse of notation in the object-oriented approach. For instance, let us consider a class *Employee* having an attribute *Employment*. Then we can define a sub-class *Engineer* of *Employee* that groups employees whose attribute value for *Employment* is equal to *Engineer*.

$$Use(org, po, pv) \leftarrow \\ Use(org, po, Purpose) \wedge Declared_purpose(po, pv)$$

We can also consider that sub-views of view *Purpose* may be associated with some specific attributes (see below for examples).

The access control policy then specifies that some roles are permitted to insert some objects in the view *Purpose* or in a sub-view of the view *Purpose*. This last case is useful to specify, for example, that a medical researcher is permitted to declare the epidemiological analysis purpose but not another purpose.

By inserting an object in the view *Purpose*, a subject will declare that another subject will perform some activity in a given context. Notice that in our model, there are two subjects involved in the process of context declaration: the subject who is declaring the context (the declarant) and the subject who takes advantage of this declaration (the recipient). The policy can specify that these two subjects must be identical. For instance, in the above example, the medical researcher may be only permitted to declare a context that applies to himself or herself. However, it is also possible that the policy specifies that these two subjects may be different, provided that these subjects satisfy some constraints. For instance, a medical researcher may be permitted to declare that his or her assistant will perform some activity in some given context. In this case, the subjects are different but the declarant must be a medical researcher and the recipient must be the medical researcher's assistant.

In our approach, we use other contexts, generally prerequisite contexts, to model this kind of constraints that associates the declarant and the recipient. In the next sub-section, we define the *Personal_purpose* context (to specify that the declarant must be equal to the recipient) and the *Assistant_purpose* context (to specify that the recipient must be a declarant's assistant).

To sum up, the definition of a user-declared context has two steps:

1. Specification of roles who are permitted to declare some given purpose.
2. Specification of roles that are permitted to perform some activities in the associated user-declared context.

We illustrate these two steps in the following sub-section. Notice also that activation of a user-declared context is often associated with provisional obligation (see section 3.6).

Example of user-declared context

Let us illustrate these two steps through the following two examples:

- Rule 1: In hospital *H1*, a user empowered in the role "medical researcher" is permitted to declare the context "epidemiological analysis" for himself or herself. In this context, this user is permitted to have an access to some statistical database.
- Rule 2: In hospital *H2*, a user empowered in the role "medical researcher" is permitted to declare the context "medical research" for one of his or her assistant. In this context, this assistant is permitted to have an access to some research database.

First step: Regarding Rule 1, we specify that subjects empowered in role *Medical_researcher* are permitted to declare the purpose *Epidemiology* that applies to themselves:

- *Permission(H1, Medical_researcher, declare, Epidemiology, Personal_purpose)*

In this *Permission*, *declare* is an activity and *Personal_purpose* is a context defined as follows:

- $\forall org, \forall s \in S, \forall \alpha \in A, \forall po \in PO,$
 $Hold(org, s, \alpha, po, Personal_purpose) \leftarrow$
 $Use(org, po, Purpose) \wedge Recipient(po, s)$

that is a subject s is in context *Personal_purpose* if there is a purpose object po having s as a recipient.

Regarding Rule 2, we specify that subjects empowered in role *Medical_researcher* are permitted to declare the purpose *Medical_research* that applies to one of their assistants:

- $Permission(H1, Medical_researcher, declare,$
 $Medical_research, Assistant_purpose)$

where *Assistant_purpose* is a context defined as follows:

- $\forall org, \forall s \in S, \forall s' \in S, \forall \alpha \in A, \forall po \in PO,$
 $Hold(org, s, \alpha, po, Assistant_purpose) \leftarrow$
 $Use(org, po, Purpose) \wedge$
 $Recipient(po, s') \wedge Assistant(s, s')$

that is a subject s is in context *Assistant_purpose* if there is a purpose object po having s' as a recipient where s' is an assistant of s (represented by the application dependent predicate $Assistant(s, s')$).

Second step: Regarding Rule 1, we specify that subjects empowered in the role *Medical_researcher* are permitted to consult objects belonging to view *Statistic_database* in the user-declared context *Epidemiology*:

- $Permission(H1, Medical_researcher, consult,$
 $Statistic_database, user_declared(Epidemiology))$

Regarding Rule 2, we specify that subjects empowered in the role *Medical_assistant* are permitted to consult objects belonging to view *Reseach_database* in the user-declared context *Medical_research*:

- $Permission(H1, Medical_assistant, consult,$
 $Reseach_database, user_declared(Medical_research))$

Urgency as a user-declared context

Managing urgency is an important requirement of medical applications. However, it is a complex problem to characterize the conditions that, when satisfied, activate the urgency context. Actually, we argue that it would not be possible to give an exhaustive specification of such conditions. And even though this would be possible, it would be quite difficult for the information system to automatically check that one of these conditions is satisfied because most of them actually depend on the physician's judgement.

This is why we suggest modelling urgency as a user-declared context. Thus, we first define a purpose value *Urgent_consultation* and we associate it with a view *Urgent_consultation* which is a sub-view of view *Purpose*. By inserting an object in this view, a physician is allowed to declare that he or she is consulting a given patient in urgency.

View *Urgent_consultation* has three attributes: *Recipient* (inherited from

view *Purpose*), *Declared_purpose* (also inherited from view *Purpose* and instantiated to the purpose value *Urgent_consultation*) and a third attribute *Admitted_patient*. If *po* is a purpose object belonging to view *Urgent_consultation*, then *Admitted_patient(po, patient)* represents the name of the patient admitted in urgency.

First step: We specify that physicians are permitted to declare that they are consulting a patient in urgency:

- $Permission(H1, Physician, declare, Urgent_consultation, Personal_purpose)$

Second step: We specify the permission that applies to a physician in the user-declared context of *Urgent_consultation*. For instance, we can consider that a physician is permitted to consult the patient's medical record:

- $Permission(H1, Physician, consult, Medical_record, user_declared(Urgent_consultation))$

However, this is probably not what we want because in this case, when declaring the context *Urgent_consultation*, a physician would get an access to *every* medical record. We may prefer to further restrict this last permission so that a physician will only get an access to the medical record of a patient admitted in urgency by this physician. In this case, we have to “customize” the user-declared context *Urgent_consultation*. This is done by considering the context *Urgency* defined as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall po \in PO, \forall patient \in S,$
 $Hold(H1, s, \alpha, o, Urgency) \leftarrow$
 $Use(H1, po, Urgent_consultation) \wedge$
 $Recipient(po, s) \wedge$
 $Admitted_patient(po, patient) \wedge$
 $Name_patient(o, patient)$

that is, in the organization *H1*, subject *s* performs action α on object *o* in the context *Urgency* if there is a purpose object *po* in view *Urgent_consultation* for a recipient whose name is *s* and for an admitted patient whose name is *patient* and *o* is an object related to this patient admitted in urgency (represented by the application dependent predicate *Name_patient(o, patient)*).

We can now specify that a physician is permitted to consult the medical record of a patient admitted in urgency by this physician:

- $Permission(H1, Physician, consult, Medical_record, Urgency)$

3.5 Prerequisite context

Principle

In many cases, a permission (or an obligation or a prohibition) is granted to a subject, but only if some specific conditions are satisfied. For instance, let us turn back to the example presented in section 2.8. This example says that a physician is permitted to consult the patient's medical record. However, a specific condition must be satisfied, that is this record corresponds to the physician's patient. This condition is modelled by a context called *Attending_physician*.

We assume that the information required to check this context, namely the set of patients attended by each physician, is stored into the *system database*. Thus, the evaluation of such a context is done by querying the database. This kind of contexts are called *prerequisite context*.

Examples of prerequisite context

We already give in the previous sections examples of context that involve in their definition prerequisite conditions: the context *Personal_office* (see section 3.3) or the contexts *Personal_purpose*, *Assistant_purpose* or *Urgency* (see section 3.4).

Let us consider another example: “In hospital *H1*, a nurse is granted the permission to consult a medical record in the context where the physician of the corresponding patient is absent”. This corresponds to the following permission:

- $Permission(H1, Nurse, consult, Medical_record, Absent_physician)$
 where the prerequisite context *Absent_physician* is expressed as follows:
- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall s' \in S, \forall patient \in S,$
 $Hold(H1, s, \alpha, o, Absent_physician) \leftarrow$
 $Use(H1, o, Medical_record) \wedge$
 $Name_patient(o, patient) \wedge$
 $Empower(H1, s', physician) \wedge$
 $Patient(s', patient) \wedge$
 $Status(s', Absent)$

that is subject *s* performs action α on object *o* in context *Absent_physician* if *o* is a medical record of some patient *patient* and the physician *s'* of this *patient* is absent (represented by the application-dependent predicate $Status(s', Absent)$).

Notice that we decide to define the context *Absent_physician* as a prerequisite context, that is it is evaluated by querying the database to check if the physician of a given patient is absent. This will be possible if the database actually stores such information.

If this is not the case, then another possibility would be to define the context *Absent_physician* as a user-declared context. For instance, the nurse may be permitted to declare this context for a given patient. Of course, the two policies will not be identical since, in this second case, the nurse will be responsible for declaring the context *Absent_physician*.

This means that the fact that a given context will be defined as a prerequisite context or as a user-declared context strongly depends on the data stored in the system database.

3.6 Provisional context

Principle

The notion of provisional *obligation* was first introduced in [35, 30]. A provisional obligation is an obligation to perform some action that applies when some subject performs another action, generally in a given context (typically when the context is user-declared). In this case, the provisional obligation is automatically triggered as a counterpart of the action performed by the subject.

We suggest modelling this notion using another type of context called *provisional context*. For this purpose, we shall first assume that each organization manages a log

file, that stores data about previous actions done by subjects in this organization. This is modelled by a view called *Log*.

Objects belonging to view *Log* correspond to entries in the log files. These objects have five attributes: *Log_actor*, *Log_action*, *Log_target*, *Log_date*, and *Log_context*. If *l* is a log file entry, then these attributes respectively represents the subject *s* (*Log_actor*(*l*, *s*)) who is performing an action α (*Log_action*(*l*, α)) on an object *o* (*Log_target*(*l*, *o*)) at a given date *d* (*Log_date*(*l*, *d*)) in a context *c* (*Log_context*(*l*, *c*)).

The reference monitor is responsible for inserting entries in the log file. When a request is accepted by the reference monitor, then this monitor will register in the log file the requested subject, action and object and the corresponding date of the request. The registered context corresponds to the security rules that were applied to accept the request. Since there may be several such security rules, this means that several associated contexts may be actually registered in the log entry.

Example of provisional context

To illustrate the approach, let us show how to model the following rule: “In the hospital *H1*, if a physician consults a given patient’s medical record in a context of urgency, then this physician has a provisional obligation to send a medical report to the attending physician of this patient”.

To model this rule, we first define a provisional context called *Urgent_consultation* as follows:

- $\forall s \in S, \forall \alpha \in A, \forall o \in O, \forall l \in O, \forall \alpha' \in A,$
 $Hold(H1, s, \alpha, o, Urgent_consultation) \leftarrow$
 $Use(H1, l, Log) \wedge Log_actor(l, s) \wedge$
 $Log_action(l, \alpha') \wedge Consider(H1, \alpha', consult) \wedge$
 $Log_context(l, Urgency)$

that is, in *H1*, subject *s* performs action α on object *o* in provisional context *Urgent_consultation* if an entry *l* was logged in view *Log* with (1) subject *s* as an actor, (2) an action α' that is considered a *consult* activity and (3) a context of *Urgency*.

We then define a view *Medical_report* having three attributes: *Addressee* (the subject who is supposed to receive the report), *Name_patient* (the patient’s name concerned by the report) and *Content* (the content of this medical report). Based on this view, we define a view *Med_report_to_attending_physician* which corresponds to the following view definition:

- $\forall rep \in O, \forall patient \in S, \forall s \in S,$
 $Use(H1, rep, Med_report_to_attending_physician) \leftarrow$
 $Use(H1, rep, Medical_report) \wedge Name_patient(rep, patient) \wedge$
 $Addessee(rep, s) \wedge Patient(s, patient)$

Using this provisional context and this view, we can then specify that a physician has a provisional obligation to send a medical report to the attending physician of this patient:

- $Obligation(H1, Physician, send,$
 $Med_report_to_attending_physician, Urgent_consultation)$

Other applications of provisional context

In [35], the authors consider two different types of provisional obligation: obligations that are triggered *after* executing a given action (see the example in the previous section) and obligation that must be fulfilled *before* executing a given action.

Let us call “before obligation” the second type of provisional obligation and let us assume that a before obligation o_1 must be fulfilled before a permission p_1 is granted. In our approach, before obligation o_1 is actually modelled as a *permission*. If a given user performs the action corresponding to this permission, then a given provisional context is activated. In this provisional context, the (provisional) permission p_1 will be granted.

The advantage of this approach is that we only need to check (past) historical data to define provisional context. Notice also that a permission may be viewed as provisional (and not only an obligation). We are currently applying this notion of provisional permission to *workflow* access control. This will be useful to model that, in a workflow, permissions are granted as the user advances in a given task.

4. Decidability and complexity

The OrBAC model is based on first order logic. The main drawback of full first order logic is that it leads to undecidable theory. Thus, we have to restrict the regulated system definition to obtain a decidable and tractable theory. One way to proceed is to consider that a regulated system corresponds to a Datalog program [45]. Datalog programs do not allow the use of functional terms. Furthermore, Datalog programs must only include both defined and safe rules. A rule is defined if every variable that appears in the conclusion also appears in the premise. A rule is safe if it only provides means to derive a finite set of new facts.

In a pure Datalog program, rules do not contain any negative literal. Pure Datalog guarantees that any access control policy will be decidable in polynomial time. However pure Datalog expressivity is very restricted.

In Datalog[¬], the negation restriction is relaxed. Negative literals are allowed but rules must be stratified. Stratifying a Datalog program consists in ordering rules so that if a rule contains a negative literal then the rule that defines this literal is computed first. A stratified Datalog[¬] program is computable in polynomial time.

We actually assume that the definition of security policies using the OrBAC model obeys the Datalog[¬] restriction except the definition of contexts through the *Hold* predicate.

More precisely, the security rules correspond to ground close facts specified using the *Permission*, *Obligation*, *Prohibition* and *Dispensation* predicates. Regarding the specifications of predicates *Empower*, *Use* and *Consider*, we also consider that they correspond to facts or to rules (corresponding to role, view or activity definitions) that respect the Datalog[¬] restrictions. We also assume that the *Hold* predicate is not used in the premises of these rules.

By contrast, we cannot assume that the definition of contexts corresponds to Datalog[¬] rules for the following reasons:

- These rules contain functional terms. In particular, the context algebra presented in

section 2.6 may be used to derive infinite functional terms by iteratively applying the functions $\&$, \oplus and $\bar{}$ to build more complex contexts.

- These rules are not always safe. For instance, temporal context definitions such as *after_time(t)* may derive an infinite set of new facts when time t is not fixed.
- These rules are not always defined. For instance, in temporal context definitions, subjects, actions and objects are respectively universally quantified over the set S , A and O but are not further constrained in the premises of the rule. This may lead to evaluate large cartesian products which is not efficient.

To solve these problems, our proposal is the following.

We first restrict the theory so that we do not attempt to evaluate any contexts but only *relevant* contexts. A context is relevant if it appears in the definition of a security rule. As a consequence, a relevant context is always of finite size and since the rules of section 2.6 for the context algebra converge to the evaluation of smaller size contexts, the process of *relevant* context evaluation terminates. This solved the first aforementioned problem.

A relevant context is always fully instantiated. For instance, we shall never attempt to evaluate *after_time(t)* where t is a variable. This solve the second problem.

Finally, in Datalog, queries are pre-computed iteratively using a bottom-up strategy until convergence to a fixed-point. As mentioned in [6], this would not be an acceptable strategy when the queries depend on function calls over the environment, for example for getting the current time to evaluate the temporal contexts. Thus, a top-down evaluation is clearly more adapted to evaluate the relevant contexts. However, the usual top-down algorithms based on SLD and used in Prolog does not guarantee termination even when a finite fixed-point exists. By contrast the SLG algorithm preserves the termination [44]. This approach is far more efficient than the bottom-up method when we have fully instantiated *Hold* predicates to evaluate. This will be generally the case when we have to decide that a given access must be accepted or denied. This solves the third aforementioned problem.

To sum up, our approach is to pre-compute the evaluation of the *Empower*, *Use* and *Consider* predicates using a bottom-up strategy. This is possible since rules defining these predicates respect the Datalog $\bar{}$ restrictions and do not include the *Hold* predicate. Then, we complete the evaluation of queries using the top-down strategy as defined in the SLG algorithm.

This hybrid strategy guarantees the decidability of query evaluation in the OrBAC model and its termination in polynomial time.

5. Related work and discussion

5.1 Related work

Several approaches have been proposed to model context management in access control and security policies. As we have seen earlier, the context makes it possible to express different kinds of constraint.

In the RBAC family models [40], RBAC₂ introduces the notion of constraints that control user-role assignment, permission-role assignment and session-role assignment constraints. Thus, RBAC₂ makes it potentially possible to establish security rules that depend

on certain context information. Even though the constraints are not modelled in the initial RBAC model, some work have been done to formalize them [2, 3].

Many existing works also attempt to extend the RBAC model to deal with access control based on users' location context [9, 13, 14, 15, 23, 27, 42, 46, 47] or temporal context [8, 32, 33]. Most of these approaches suggest combining the concept of role with spatial or temporal contextual conditions to obtain "contextual" roles.

Hence, [14] offers to broaden the role concept to the security requests environment. In the generalized model [15], the environment context is specified through a new type of role called the *environment role*. It is thus possible to express the constraints which are related to time intervals and trusted locations. Moreover the environment role hierarchy makes the contextual information management easier. In [27], the authors discuss the identification of role with respect to location and assign the permissions in specific sessions considering the role and location with emphasis on roles belonging to some organization. The GEO-RBAC model [9], which possibly is the most expressive location based RBAC model, assumes that users can use a role only from a particular location, and the role and its associated permissions are predefined for that location. In [46], a distributed RBAC approach is presented which is based on *logical* location-dependant conditions. An extensive set of predicates is defined to model the network and computer characteristics and used to specify the scheduling of distributed on-line processes for automated location-dependant negotiating procedures.

In [8], the authors propose another extension of the RBAC model, called Temporal Role-Based Access Control (TRBAC). This work does not aim to express temporal conditions over the time of the action corresponding to the security request but offer means to activate roles periodically or thanks to a trigger. The TRBAC model is then refined in GTRBAC [33] where temporal constraints and locations are separated with locations being structured in physical, logical and hybrid location hierarchies.

The concept of *purpose* was also suggested in several privacy models for instance the Privacy Model [22]. In order to enforce privacy, it should be checked whether the purpose of the task, currently performed by the user who wants to access personal data, corresponds to the purpose for which that personal data were obtained.

In several works related to context, the context corresponds to the ongoing activity in which the permissions are requested. In this case, controlling the workflow makes it possible to "filter" the privileges granted to users. [24, 25] provide a model based on TMAC, called C-TMAC, in which users obtain permissions according to their role and the team in which they are involved. A context which defines the time, the location, etc, is associated to the team. These attributes are used to reduce permissions granted to each role. This model is particularly adapted to collaborative environments.

Through C-TMAC, contexts are used to take into account the need-to-know requirement, and the notion of just-in-time permission activation. A great number of works such as [29] are based on this idea.

The expression of a workflow environment through the context can also be considered as a means to respond to the least privilege principle. The option of expressing contexts is useful in the case of business and transaction activities. In such areas, the access control decision depends on specific sequences of events [1, 5, 10, 11]. In [1], the Workflow

Authorization Model (WAM) is defined that is capable of specifying authorizations in such a way that subjects gain access to required objects only during the execution of the task. In [11], the authors describe a context-sensitive access control model in which the rights are granted according to the actual task. The multiple tasks are defined in a global process. The context is activated through reference to this process definition.

Contextual conditions are also useful to specify usage control policies as suggested in the UCON model [39]. In this model, pre conditions are checked to grant the access and on-going conditions are checked during the access. Moreover, it is also possible to specify environmental conditions that do not depend on the subject which is requesting the access, for instance temporal conditions. The taxonomy of contexts we suggest in this paper may help in structuring the different conditions suggested in UCON.

If an access or usage control model must allow the expression of contexts, it is also important to be able to evaluate this context. [29] shows how to apply a context-dependent access control mechanism on a commercial platform. Through the Antigone Condition Framework (ACF), [37] provides means to specify, implement and evaluate the context which is viewed as a set of external conditions. Evaluation of contextual conditions associated with usage control requirements is discussed in [41].

5.2 Discussion

The different existing models make it possible to capture information related to time, or to the user location, or to the team to which the user belong, or to the current workflow, etc. Even though all these models are useful, they do not provide means to express a large number of different contexts within a single framework. It was precisely one of our objectives in this paper to define such an homogeneous unified framework.

Moreover, we argue that the notion of “contextual” role that was suggested in several models generally corresponds to artificial roles and sometimes to misleading roles. For instance, let us consider a role *urgency physician*. This is a new role corresponding to “special” physicians that work in a service of urgency. It is clear that such a role *urgency physician* is distinct from a “standard” physician working in a context of urgency. In particular, the permissions assigned to this *urgency physician* are distinct from those assigned to a standard physician (even if this standard physician is working in a context of urgency). Moreover, an *urgency physician* does not perform all his or her activities in a context of urgency.

Thus, it is important to clearly separate the concepts of role and context. In our approach, a context is actually not attached to the role but to the security rules. Thus we can assign, to the role *urgency physician*, security rules that apply in contexts different from the one of urgency (when the role *urgency physician* does not perform an activity in a context of urgency).

Our approach also clearly separates the concept of organization and the spatial context of being located in an organization. Thus, there is a clear distinction between a subject being empowered in a given role in a given organization (represented by the predicate *Empower*) and a subject being located in a given organization (represented by the predicate *Is_located*). We also separate the security rules defined by some organization and

that apply when a subject is located in the spatial context of the same or another organization (represented by $location(org)$).

For example, let us consider two hospitals $H1$ and $H2$. It is then possible to consider security rules defined in organization $H1$ and in the spatial context $location(H1)$ (which apply to subjects empowered in hospital $H1$ when they are physically located in hospital $H1$) or security rules defined in organization $H1$ and in the spatial context $location(H2)$ (which apply to subjects empowered in hospital $H1$ when they are physically located in hospital $H2$). Similarly, we can also consider other security rules defined in organization $H2$ and in the spatial context $location(H1)$ or $location(H2)$.

By contrast, a “combined” role such as $H1_physician$ is ambiguous since it may be interpreted as a “contextual” role (corresponding to physician *located* in hospital $H1$) or an “organizational” role (corresponding to physician *empowered* in hospital $H1$).

We also introduce the concept of *user declared* context to model that a user specifies that he or she will perform some activities in a given purpose. The user declared contexts are also useful to model contexts that are difficult to describe using environmental conditions such as the *urgency* context.

The last type of concept we consider in this paper is the *provisional* context. It is used to model security rules whose activation depends on the history of previously actions performed by the user. We present the basic principles to model this kind of context in our formalism. We are currently refining this approach to define a model for workflow security but this is out of the scope of our purpose to develop this aspect here.

This paper has presented a formalism to express the main types of context that are useful to specify a security policy. We do not claim that this taxonomy is complete and that other types of context could not be considered. In particular and as suggested in [21], the concept of *intrusion* context can be introduced to specify security rules to be activated as a response to an intrusion. The approach is based on an intrusion detection framework that launches alerts when an attack is detected. As suggested in [21], the attack described in the alert is mapped onto an intrusion context so that security rules associated with this context is activated when this attack is detected.

6. Conclusion

In this paper, we present a model that includes an explicit expression of context and show how to use it to specify dynamic and flexible access control rules. We suggest a taxonomy of different types of context and model them in the OrBAC model. Starting from elementary contexts, we also define conjunctive, disjunctive and negative contexts.

To control activation of a given context, the information system must store different data: temporal data to manage temporal context, user environment and system architecture representation for spatial context, and application data stored in the information base to define prerequisite context.

We also show that it is sometimes not possible to express all the possible conditions required to activate some contexts. For instance, if we consider the medical context of *urgency*, there are many different possibilities so that it is actually impossible to provide an exhaustive definition of such a context. In this case, we suggest defining the urgency

context as a *user-declared* context: this is the responsibility of some authorized user to declare that this context is activated.

Thus, to activate a user-declared context, the user must be authorized to declare some objective (or purpose) of his or her activity. This is modelled by views, a given user-declared context being activated by inserting a given object in this view.

Finally, we define provisional context. This is used to model permissions, obligations or prohibitions that depend on previous actions performed by the user. To control activation of provisional context, the information system must store historical data of what happens in the system. Information systems generally provide such historical data through audit trail. Provisional obligation was already suggested in previous research work. Provisional permission may be also useful to model situations where users obtain permissions as their work proceeds. Similarly, provisional prohibition is another useful context to model situations where the user's previous activity leads to prohibition.

There were several other proposals to model contexts within an access control model but this is the first time that all the different contexts are expressed within a unique homogeneous framework. We are currently further investigating the notion of provisional context, in particular to model management of rights in workflow system. We are also applying this model in the framework of relational database administration.

Acknowledgements

This work was partially funded by the RNRT project Politess.

References

- [1] N. Adam, V. Atluri, and W.-K. Huang. Modelling and Analysis of Workflows Using Petri Nets. 10(2), 1998.
- [2] Gail-Joon Ahn and Ravi Sandhu. Role-based Authorization Constraints Specification. *ACM Transactions on Information and Systems Security*, 3(4), 2000.
- [3] Gail-Joon Ahn and Michael. E. Shin. Role-based Authorization Constraints Specification Using Object Constraint Language. In *10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)*, Massachusetts, USA, 2001.
- [4] M. A. Al-Kahtani and R. Sandhu. A Model for Attribute-Based User-Role Assignment. In *18th Annual Computer Security Applications Conference (ACSAC '02)*, Las Vegas, Nevada, December 2002.
- [5] V. Atluri and W.-K. Huang. An Authorization Model for Workflows. In *ES-ORICS'96*, volume 1146. LNCS, 1996.
- [6] M. Becker and P. Sewell. Cassandra: Flexible Trust Management, Applied to Electronic Health Records. In *The computer security foundations workshop (CSFW)*, Pacific Grove, CA, June 2004.
- [7] D. E. Bell and L. J. La Padula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, MTR-2997, Rev. 1, MITRE Corporation, Bedford, MA, March 1976.

- [8] E. Bertino, P. A. Bonatti, and E. Ferrari. A Temporal role-based access control model. *ACM transactions on information and System Security*, 2001.
- [9] E. Bertino, E. Catania, M. Damiani, and P. Persasca. GEO-RBAC: A Spatially Aware RBAC. In *10th ACM Symposium on Access Control Models and Technologies (SACMAT 2005)*, Chantilly, Virginia, USA, May 2001.
- [10] E. Bertino, E. Ferrari, and V. Atluri. An Authorization Model for Supporting the Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Transactions on Information and System Security*, 2(1), 1999.
- [11] D. G. Cholewka, R. A. Botha, and J. H. P. Eloff. A Context-sensitive Access Control Model and Prototype Implementation. In *IFIP TC 11 16th Annual Working Conference on Information Security*, Beijing, China, August 2000.
- [12] Open GIS Consortium. Open GIS Simple Features Specification for SQL. Revision 1.1. 1999.
- [13] A. Corradi, R. Montanari, and D. Tibaldi. Context-Based Access Control in Ubiquitous Environments. In *3rd IEEE International Symposium on Network Computing and Applications (NCA)*, August 2004.
- [14] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. Securing context-aware applications using environment roles. In *6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, Chantilly, Virginia, USA, May 2001.
- [15] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *Proceedings of the 23rd National Information Systems Security Conference, (NISSC)*, Baltimore, MD, USA, October 2000.
- [16] F. Cuppens, N. Cuppens-Boulahia, and M. Ben Ghorbel. High-level conflict management strategies in advanced access control models. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 186, July 2007.
- [17] F. Cuppens, N. Cuppens-Boulahia, and A. Miège. Inheritance hierarchies in the Or-BAC Model and Application in a network environment. In *2nd Foundation of Computer Security Workshop*, Turku, Finlande, July 2004.
- [18] F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Nomad: A Security Model with Non Atomic Actions and Deadlines. In *The computer security foundations workshop (CSFW)*, Aix en Provence, France, 2005.
- [19] F. Cuppens and A. Miège. Modelling Contexts in the Or-BAC Model. In *19th Annual Computer Security Applications Conference (ACSAC '03)*, 2003.
- [20] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *International Workshop, Policies for Distributed Systems and Networks (Policy 2001)*, Bristol, UK, January 2001.
- [21] H. Debar, Y. Thomas, N. Cuppens-Boulahia, and F. Cuppens. Using contextual security policies for threat response. In *Detection of Intrusions & Malware, and Vulnerability Assessment (DIMVA)*, 2006.
- [22] S. Fischer-Hubner and A. Ott. From a Formal Privacy Model to its Implementation. In *Proceedings of the 21st National Information Systems Security Conference*, 1998.
- [23] S. Fu and C.-Z. Xu. A Coordinated Spatio-Temporal Access Control Model for Mobile Computing in Coalition Environments. In *19th IEEE International Parallel*

- and *Distributed Processing Symposium*, April 2005.
- [24] C. Georgiadis, I. Mavridis, and G. Pangalos. Context and Role Based Hybrid Access Control for Collaborative Environments. In *Proceedings of the Fifth Nordic Workshop on Secure IT Systems - Encouraging Co-operation (NORDSEC 2000)*, Reykjavik, Iceland, October 2000.
 - [25] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas. Flexible team-based access control using contexts. In *6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*, Chantilly, Virginia, USA, May 2001.
 - [26] Joseph Halpern and Vicky Weissman. Using First-Order Logic to Reason about Policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, 2003.
 - [27] F. Hansen and V. Oleshchuk. Spatial Role-Based Access Control Model for Wireless Networks. In *IEEE 58th Vehicular Technology Conference, VTC 2003-Fall*, volume 3, October 2003.
 - [28] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *CACM*, 19(8):461–471, August 1976.
 - [29] R. Holbein, O. Morger, U. Nitsche, and S. Teufel. Realization of a Context-Dependent Access Control Mechanism on a Commercial Platform. In *IFIP 14th International Conference on Information Security (IFIP/Sec'98)*, Vienna-Budapest, Austria-Hungary, August 31 - September 4 1998.
 - [30] S. Jajodia, M. Kudo, and V.S. Subrahmanian. Provisional Authorizations. In A. Ghosh, editor, *E-commerce Security and Privacy*, pages 133–159. Kluwer Academic Publishers, 2001.
 - [31] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, California, USA, 1997.
 - [32] J. B. D. Joshi, E. Bertino, and A. Ghafoor. Analysis of Expressiveness and Design Issues for a Temporal Role Based Access Control Model. 2005.
 - [33] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor. Generalized Temporal Role-Based Access Control Model. 17(1):4–23, January 2005.
 - [34] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, Lake Como, Italy, June 2003.
 - [35] M. Kudo and S. Hada. XML Document Security based on Provisional Authorization. In *Proceedings of the 7th ACM Conference on Computer and Communication Security (CCS 2000)*, Athens, Greece, 2000.
 - [36] Markus G. Kuhn. Optical Time-Domain Eavesdropping Risks of CRT Displays. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Oakland, California, USA, 2002.
 - [37] Patrick McDaniel. On Context in Authorization Policy. In *Proceedings of the 8th ACM Symposium On Access Control Models and Technologies (SACMAT 2003)*, Como, Italy, June 2003.

- [38] F. Pan and J. Hobbs. Time in OWL-S. In *AAAI Spring Symposium Series on Semantic Web Services*, Palo Alto, CA, 2004.
- [39] J. Park and R. S. Sandhu. The $UCON_{ABC}$ usage control model. *ACM Transactions on Information and System Security*, 7(1), 2004.
- [40] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.
- [41] T. Sans, F. Cuppens, and N. Cuppens-Boulahia. A Flexible and Distributed Architecture to Enforce Dynamic Access Control. In *21st IFIP TC-11 International Information Security Conference (SEC 2006)*, Karlstad, Sweden, May 2006.
- [42] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM transactions on information and System Security*, 7(3):392–427, 2004.
- [43] Roshan K. Thomas. TMAC: A primitive for Applying RBAC in collaborative environment. In *2nd ACM, Workshop on RBAC*, pages 13–19, FairFax, Virginia, USA, 1997.
- [44] D. Toman. Memoing Evaluation for Constraint Extensions of Datalog. *Constraints*, 2(3/4):337–359, 1997.
- [45] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.
- [46] H. Wedde and M. Lischka. Role-Based Access Control in Ambient and Remote Space. In *9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, Yorktown Heights, New York, USA, June 2004.
- [47] G. Zhang and M. Parashar. Dynamic Context-aware Access Control for Grid Applications. In *4th International Wrokshop on Grid Computing*, November 2003.