Formal contextual security model for pervasive computing applications

Yehia ElRakaiby, Frederic Cuppens, Nora Cuppens-Boulahia
TELECOM-Bretagne
35510, Cesson Sevigne, France
{yehia.elrakaiby, frederic.cuppens, nora.cuppens*g*@telecom-bretagne.eu}

ABSTRACT: *In this paper, we present a formal contextual security model for pervasive computing applications. This model is developed by addressing the required featurs such as - support of authorization and obligation policies, monitoring and dynamic revocation of access rights, support of personalized security rule contexts, and support of collaborative applications. Besides we design the model in a way to be logic-based. The mdoel is thus elegant as it enables the use of formal policy conflict and dynamic system analysis techniques.*

Keywords: Pervasive computing, Contextual security model, Access rights, Logic-based security model

Received: 11 June 2010, Revised 10 July 2010, Accepted 19 July 2010

1. Introduction

Traditional access control models like [8] considered static permissions. More recent models like [1, 10, 15, 4] integrated contextual conditions. Both these systems simply provided a yes/no answer to access requests.

New interactive smart environments introduced new challenges and requirements. In particular, in these highly dynamic environments, one must consider usage controls [13]. Usage controls are conditions which must be satisfied before, while or after access. Usage controls may take the form of obligations. For instance, consider the two requirements “Before a subject is granted access to the research lab, the subject must be properly authenticated” and “When a patient is admitted to the emergency room, the patient should be examined by one of the doctors within 10 minutes.” These two obligations represent a pre- and postusage control requirements respectively.

Usage controls may also take the form of state conditions which must hold while access occurs. For instance, consider the permission “a professor may control the projector while there is an ongoing lecture”. The enforcement of this permission requires that the professor’s access right to the projector be revoked whenever the lecture is ended.

To enable the specification of fine-grained contextual security rules, personalized rule contexts need to be supported. For instance, consider the permission “a patient’s secret files may only be consulted by the doctor assigned to the patient during the day”. This permission specifies two constraints: a particular relationship between the security rule subject (the doctor) and object (the patient), and a state constraint (by day). A contextual security model should enable the specification of such personalized state constraints.

In smart environments, there is often a need to support collaborative applications. For instance, consider the permission “During a meeting application, a subject may only display on the screen information that may be viewed by all subjects present in the meeting.” The support of this type of applications requires that authorization policies be sensitive to change in state conditions and in subjects’ access rights.

In this paper, we present a security policy management framework which satisfies the above requirements. The framework extends contextual security policies in the *OrBAC* model [6] to support permission monitoring and revocation. We also integrate obligations into the framework. The proposed framework is formalized. This enables the use of advanced logic-

based policy analysis techniques [2].

The paper is organized as follows. Section 2 presents a motivating example. Basic concepts are introduced in Section 3. Section 4 presents our access control policy language. The dynamic management of authorization policies is discussed in Section 5. Section 6 presents our context language. The obligation policy language is presented in Section 7. Section 8 gives an application example. Finally, Section 9 discusses related works and Section 10 concludes the paper.

2. Motivating Example

To motivate our work, we consider the following security policy of an intelligent campus. Some of the examples are inspired from [14].

p_1: A professor may start a lecture only when there is more than 5 students present in the classroom.

p_2: When a professor starts a *lecture*, only the professor is allowed to control the projector.

p_3: During a lecture, Students in the classroom may only read and write on the white board.

p_4: During a meeting, only information that all present subjects are allowed to consult may be displayed on the screen.

o_1: When the professor starts a lecture, the professor should turn on the video projector within 5 minutes.

The correct enforcement of the policy above requires the support of authorization and obligation policies, the dynamic reconfiguration of authorization policies and permission revocation, and the support of personalized contexts, *e.g.* to activate the permission to use the projector only for the professor who initiated the lecture application.

3. Basic Concepts

3.1 System Object & State Representation

We consider a sorted first-order language which includes finite sorts for subjects *S*, objects *O*, actions *A*, contexts *C*, rule identifiers $\mathcal{N}$, predicate symbols *Q* and variables $\mathcal{V}$. To enable the specification of security rules for groups of subjects, actions and objects, we also consider the sorts roles ($\mathcal{R}$), activities ($\mathcal{A}$) and views ($\mathcal{V}$), respectively. Constants and variables are terms of the language. A rule is a formula written as $A \leftarrow B_1,...,B_n$ where $A,B_1,...,B_n$ are atoms. An atom is a formula $Q(t_1,...,t_i)$ where *Q* is a predicate symbol and each t_i is a term. Variable free atoms are facts. Facts, also called fluents, may change over time.

3.2 The *OrBAC* model

The Organization-Based Access Control Model is a logic-based contextual security model. In this paper, we consider the following *OrBAC* relations[1] .

- ***Empower*** is a predicate over domains $S \times \mathcal{R}$. If *s* is a subject and *r* is a role, then *Empower(s, r)* means that subject *s* is empowered into role *r*.

- ***Use*** is a predicate over domains $O \times \mathcal{V}$. If *o* is an object and *v* is a view, then *Use(o, v)* means that object *o* is used in view *v*.

- ***Consider*** is a predicate over domains $A \times \mathcal{A}$. If α is an action and *a* is an activity, then *Consider(α ,a)* means that action α implements activity *a*.

- ***Permission*** is a predicate over the domains $N \times RS \times AA \times OV \times C$ where $SR = \mathcal{R} \cup S$, $AA = \mathcal{A} \cup A$ and $OV = \mathcal{V} \cup O$. A relation *Permission(n, sr, aa, ov, ctx)* specifies that permission *n* states that the subject/role *sr* may take the action/activity *aa* on the object/view *ov* in the context *ctx*. A fact *Permission* represents an *abstract* system permission.

- ***Permitted*** is a predicate over the domains $N \times S \times A \times O \times C$. A relation *Permitted(n, s,α ,o,ctx)* denotes that permission *n currently* authorizes subject *s* to perform action α on object *o* until the context *ctx* ends. A fact *Permitted* represents a *concrete* permission.

- ***Hold*** Security policies specify whether a subject *S* is permitted, prohibited, obliged or dispensed to take some action *A* on some object *O*. The specification of conditions over the security rule triple *(S,A,O)* and on the system state is made using the predicate *Hold* defined over the domains $S \times A \times O \times C$. A relation *Hold(s,$\alpha$, o, c)* means that the set of conditions identified by

[1]In our framework, security policies are sets of permissions, prohibitions, obligations and dispensations. However, in this paper, only permissions and obligations are considered.

c holds for the subject *s* taking the action α on the object *o*. The specification of contextual conditions using the *Hold* predicate is discussed in Section 4. The obligation language will be introduced in Section 7.

3.3 Dynamic System Modeling ($\mathcal{L}_{active}$)

To formalize access and usage control policies management operations, we use the language $\mathcal{L}_{active}$. $\mathcal{L}_{active}$ enables the description of dynamic systems. It borrows its concepts from action specification languages. A translation of the language into logical programs is presented in [3].

The alphabet of $\mathcal{L}_{active}$ consists of four sorts: (1) Fluents: are time-varying propositions representing the system state. (2) Actions: represent possible actions in the system. Actions update the state by adding and removing fluents to and from the state. (3) Events: are used to specify state conditions at which policy management operations are needed. (4) Rule Names: unique identifiers of Event Condition Action (*ECA*) rules. *ECA* rules, also called *active* rules, are used to initiate policy management operations when events are detected. An *ECA* rule states that when the *event* occurs and if *conditions* are true, then *actions* are executed.

The language has the following three propositions:

(**EL**) $a(\overline{X})$ **causes** $f(\overline{Y})$ **if** $p_1(\overline{X_1}),...,p_i(\overline{X_i})$

(**ED**) $e(\overline{Y})$ **after** $a(\overline{X})$ **if** $p1(\overline{X1}),...,p_i(\overline{X_i})$

(**AR**) $r(\overline{X_r})$: $e(\overline{X})$ **initiates** $[\alpha]$ **if** $p_1(X_1),...,p_i(\overline{X_i})$

Where the symbols $f,p_1,...,p_i$ are fluent symbols, a is an action symbol, $e_1,...,e_i$ are event symbols and r is an active rule identifier. An effect law proposition (**EL**) states that the execution of $a(X)$ in a state where the fluents $p_1(\overline{X_1}),...,p_i(\overline{X_i})$ are true causes $f(\overline{Y})$ to be true in the next state. An event definition proposition (**ED**) states that if the conditions $p_1(\overline{X_1}),...,p_i(\overline{X_i})$ are true in the state following the execution of the action $a(\overline{X})$, then event $e(\overline{Y})$ is produced. An active rule proposition (**AR**) states that every new detection of the event $e(\overline{X})$ initiates the execution of the sequence of actions $[\alpha]$ if the rule conditions are true.

The operational semantics of $\mathcal{L}_{active}$ defines a transition function which given a state and a (possibly empty) sequence of actions produces a new state as follows. Actions in the input sequence are processed successively. For every action, effect laws are evaluated and the fluent state is updated. If after the execution of the action, conditions in some event definition are true, the event is generated. The newly generated events trigger active rules. Identifiers of these triggered rules are added to the triggered rules set. When the last action in the input sequence is evaluated, if the triggered rules set is not empty, an action selection function selects the sequence of actions appearing in one of the rules in the triggered rules set to process. The state stops evolving after the processing of all the actions in an input sequence if the triggered rule set is empty. One of the main advantages of the language is that, due to its simplicity, it is amenable to efficient implementation.

4. Access Control Policy Specification

Access control requirements are specified using the *OrBAC* policy language. For instance, the permission *p*1 in Section 2 is specified as follows:

Permission(p_1, professors, start, lecture, more_Than_5_Students)

The context *more_Than_5_Students* is specified using the predicate *Hold(S,A,O,Ctx* as follows:

Hold(S,_,_,more_Than_5_Students) $\leftarrow$ *Location(S, classroom), Nb_Of_Students_In_Classrrom(N,classroom), N>5*

The rule above specifies that the context *more_Than_5_Students* should hold for a subject if the subject is in the classroom

and there is more than 5 students in the classroom. The rule above defines *more_Than_5_Students* in terms of conditions on the fluent. Therefore, it is called a *state context rule*. The context *more_Than_5_Students* is called a state context. State context rules enable the specification of many types of contextual conditions [6], *e.g.* spatial, temporal, etc.

5. Dynamic Access Policy Management

The policy in Section 4 specifies contextual permissions which enable the system to provide a yes/no answer to access requests. However, it does not enable the monitoring and revocation of permissions. In this section, we extend the model to support permission monitoring and revocation.

Our proposal consists of the following. First, we describe the system dynamic behavior using $\mathcal{L}_{active}$ effect laws, *i.e.* by specifying the effects of action occurrences on the state fluents. Using this description and state context rules, we derive two types of $\mathcal{L}_{active}$ event definitions. Events of the first type have the form $Hold_e(S,A,O,\textbf{\textit{start}}(Ctx))$. They specify the conditions at which state contexts should begin to hold. Events of the second type have the from $Holde(S,A,O,\textbf{\textit{end}}(Ctx))$. They, on the other hand, specify the conditions at which state contexts should seize to hold. These events are then used to update the applied access policy using active rules.

5.1 Dynamic State Description

The dynamic behavior of the system is described using $\mathcal{L}_{active}$ effect law propositions presented in Section 3.3. Effect laws specify the effect of the occurrence of actions on the state fluents. For instance, effects of the actions *enter* and *exit* on the fluents *Location* and *Nb Of Students* are specified as follows.

Do(*S, enter, classroom*)
causes ¬ *Nb_Students_In_Classroom*(*N, classroom*),
Nb_Students_In_Classroom(*N*+1, *classroom*)
if *Empower*(*Sm students*),
Nb_Students_In_Classroom(*N, classroom*)

Do(*S, exit, classroom*)
causes ¬ *Nb_Students_In_Classroom*(*N, classroom*),
Nb_Students_In_Classroom(*N*-1, *classroom*)
if *Empower*(*S, students*),
Nb_Students_In_Classroom(*N, classroom*)

Do(*S, enter, classroom*)
causes *Location*(*S, classroom*)

Do(*S, exit, classroom*)
causes ¬ *Location*(*S, classroom*)
if *Location*(*S, classroom*)

5.2 Deriving State Context Start and End Conditions

From the dynamic state description and state context rules, we are able to derive using algorithm 1 the conditions at which state contexts are activated and deactivated. Before presenting the algorithm, we introduce the following definitions.

Definition 1: An action *A* is said to initiate a fluent *F* if there exists an effect law "*A* causes *F* if $p_1,...,p_n$". An action *A* is said to terminate a fluent *F* if there exists an effect law "*A* causes ¬ *F* if $p_1,...,p_n$".

Definition 2: A fluent *F* is said to be positively de- fined in the context definition of *Ctx* if *F* positively appears in the conditions of the state context rule of *Ctx*, *i.e.* "*Hold*(*S, A,O,Ctx*) ⟵...,*F*,...". A fluent *F* is said to be negatively defined in the context definition of *Ctx* if ¬ *F* appears in the conditions of the state context rule of *Ctx*, *i.e.* "*Hold*(*S,A,O,Ctx*) ⟵..., ¬ *F*,...".

Definition 3: An action *A* is said to be an initiator of the context *Ctx* if *A* is an initiator of *F* and *F* is positively defined in the

context definition of *Ctx* or if *A* is a terminator of *F* and *F* is negatively defined in the definition of *Ctx*. Similarly, an action *A* is said to be a terminator of a context *Ctx* if *A* is a terminator of *F* and *F* is positively defined in the context definition of *Ctx* or if *A* is an initiator of *F* and *F* is negatively defined in the definition of *Ctx*.

Algorithm 1[2], given state context rules and effect laws, returns the event definitions necessary to monitor the activation and deactivation of state contexts as follows. For every action *A* that is an initiator of the state context *Ctx*, the algorithm returns an event definition which states that the event context ***start****(Ctx)* should be detected after the occurrence of *A* if *Ctx* does not hold in the state and conditions in the state context rule of *Ctx* are true. If *A* is, on the other hand, a terminator of *Ctx*, the algorithm returns an event definition which states that the event context ***end****(Ctx)* should be detected after the occurrence of *A* if *Ctx* holds in the state and conditions of its state context rule are no longer true.

Algorithm 1 Derive_Event_Contexts_For_State_Contexts

In: State Context Rules, Effect Laws
Out: Event Context Definitions

1: Event-Def = ∅
2: **for all** State Context Rule { Hold(Ctx) ← p_1,...,p_n } **do**
3: **for all** Condition p_i in the Rule Conditions **do**
4: **if** (p_i is positively defined and action A initiates p_i) ∨ (p_i is negatively defined and A terminates p_i) **then**
5: Event-Def = Event-Def ∪ { $Hold_e$(start(Ctx)) **After** A **If** ¬Hold(Ctx),p_1,..., p_n}
6: **if** (p_i is negatively defined and A initiates p_i) ∨ (p_i is positively defined and A terminates p_i) **then**
7: Event-Def = Event-Def ∪ {$Hold_e$(end(Ctx)) **After** A **If** Hold(Ctx),¬(p_1,...,p_n)}
8: **return** Event-Def

For instance, the execution of algorithm 1 returns event definitions of the following form for the state-based contexts and effect laws specified in Section 4.

$Hold_e$(*S*,_ ,_ ,**start**(*more_Than*_5_*Students*))
after *Do*(*S', enter, classroom*)
if ¬ *Hold*(*S*,_ ,_ ,*more_Than*_5_*Students*),
Location(*S, classroom*)*;*
Nb Of Students In Classroom(*N, classroom*),*N* > 5

$Hold_e$(*S*,_ ,_ ,end(*more_Than*_5_*Students*))
after *Do*(*S', exit, classroom*)
if *Hold*(*S*,_ ,_ ,*more Than* 5 *Students*),
¬ (*Location*(*S, classroom*),
Nb Of Students In Classroom(*N, classroom*),*N* > 5)

The first event definition states that after the execution of the action *Do(S',enter,classroom)*, if the number of students exceeds 5 and that the state context *Hold(S, , ,classroom)* does not hold for a subject in the classroom, then this state context should be started. Note that in our framework, state contexts are updated in the state following action execution.

5.3 Dynamic Management of State Contexts

When the event *$Hold_e$(S,A,O,**start**(Ctx))* is detected, the fluent *Hold(S,A,O,Ctx)* is inserted into the state to mark that the context *Ctx* now holds for the triple *(S,A,O)* using the active rule *context Activation* below. An active rule is similarly specified to remove the fluent *Hold(S,A,O,Ctx)* from the state when the end of the state context is detected.

[2]To simplify the notation, we write *A* to denote an action *Do(S',A',O')*, *Ctx* to denote a context *Hold(S,A,O,Ctx)* and *F* to denote a state fluent.

context_Activation : $Hold_e$(*S, A,O,***start**(*Ctx*))
initiates *Activate_Context*(*S, A ,O,Ctx*)

Activate_Context(*S, A,O,Ctx*)
causes *Hold*(*S, A,O,Ctx*)

5.4 Permission Activation and Deactivation
The following two active rules update the authorization policy after the activation/deactivation of state contexts.

activate_Permission : $Hold_e$(*S, A,O,***start**(*Ctx*))
initiates *Insert_Permitted*(*N, S, A,O,Ctx*)
if *Permission*(*N,R, Act, V,Ctx*),
Empower'(*S, SR*),*Consider'*(*A,AA*),*Use'*(*O,OV*)

deactivate_Permission : $Hold_e$(*S, A,O,* **end**(*Ctx*))
initiates *Remove_Permitted*(*N, S, A,O,Ctx*)
if *Permitted*(*N, S, A,O,Ctx*)

The condition *Empower'(S, SR)* specifies that *S* should either (1) be empowered into the role *SR* or (2) be the subject *SR*. The conditions *Consider'(A,AA)* and *Use'(O,OV)* are similarly defined. Formally, *Empower'(S, SR)* is speci- fied as follows:

Empower'(*S, SR*) $\leftarrow$ *Role*(*SR*),*Empower*(*S, R*)
Empower'(*S, S*) $\leftarrow$ *Subject*(*S*)

The active rule *activate Permission* above simply adds concrete permissions to the state whenever the contexts of abstract permissions are started. Concrete permissions have the form *Permitted(N,S,A,O,Ctx)*. A concrete permission denotes that rule *N* authorizes the subject *S* to take the action *A* on the object *O* until the end of the context *Ctx*. Concrete permissions remain in the state until their context (*Ctx*) ends. At this moment, they are removed from the state by the active rule *Deactivate_Permission*.

6. Context Language

Context composition enables the construction of powerful contextual conditions in security rules. We support the composition of contexts using the logical operators conjunction (Λ), disjunction ($\vee$) and negation ($\neg$). This allows the expression of more sophisticated contextual conditions, *e.g.* the contexts *(more_Than_5_Students _ campus Director Present)* and (*low_Noise_ Level*Λ *low_CPU Load*).

The activation and deactivation of composed contexts are obtained using the following context detection rules. These rules are applied only for the finite set of composed contexts which are used in the policy. This ensures that composed event detection after action occurrences always terminates.

In the following, we only present the detection conditions of the start of composed contexts. The detection conditions of their end are obtained by simply replacing the start term by the end term.

The conjunction (Λ) The start of a conjunction ($Ctx_1 \Lambda\ Ctx_2$) is detected if the start of one of the contexts in the conjunction is detected while the other context holds in the state. This is represented as follows.

start(*Ctx*1 & *Ctx*2) $\leftrightarrow$
(*Ctx*2, *start*(*Ctx*1)) $\vee$ (*Ctx*1, *start*(*Ctx*2))

The disjunction ($\vee$) The start of a disjunction (*Ctx*1 $\vee$ *Ctx*2) is detected if the start of one of the contexts in the disjunction is detected while the other context does not hold. This is represented as follows.

start(*Ctx*1 $\vee$ *Ctx*2) $\leftrightarrow$
($\neg$ *Ctx*2, *start*(*Ctx*1)) $\vee$ ($\neg$ *Ctx*1, *start*(*Ctx*2))

Negation ($\neg$) The detection conditions of the start of a negated context are the detection conditions of its end.

$start(\neg Ctx) \leftrightarrow end(Ctx)$

7. Obligation Policies

We consider obligation policies which are closed ground facts of the following form:

$$Obligation(N, SR, AA, OV, Ctx, Ctx_v)$$

Where N is a unique security rule identifier, SR is a subject or a role, AA is an action or an activity, OV is an object or a view, Ctx and Ctx_v are contexts.

The context Ctx is called the *obligation context* and it specifies when the obligation holds (is active). More precisely, an obligations is activated when this context is started and is deactivated if this context is ended while the obligation holds. Obligations are also associated with a violation context (Ctx_v) which, if started while the obligation holds, the obligation is violated. An obligation seizes to hold if it is fulfilled. For further details on the obligation language, the reader is referred to [7]. For instance, the obligation o_1 in Section 2 may be specified as follows:

$Obligation(o_1, professors, turn\ On, projector,$
$lecture\_Initiated\_By\_Professor, delay(5.minutes))$

The context *lecture Initiated By Professor* may be speci- fied using a state context rule. It may also be specified using event context rules as follows.

$Hold_e(S,\_,\_,start(lecture\_Initiated\_By\_Professor))$
after $Do(S, start, lecture)$
if $Empower(S, professors)$

$Hold_e(S,\_,\_,end(lecture\_Initiated\_By\_Professor))$
after $Do(S, end, lecture)$
if $Empower(S, professors)$

The context *delay*(*Nb.TimeUnit*) is a context that is detected *Nb* time units after the obligation is activated and represents a relative temporal deadline.

Thus, the obligation above specifies that when a professor starts a lecture, the professor should turn on the projector within 5 minutes. However, the obligation is deactivated if the lecture is ended before its deadline.

7.1 Obligation Activation and Deactivation

Obligations are managed as follows. When the obligation context is started, a concrete obligation *Obliged(N,S,A, O,Ctx,Ctx$_v$)* is added to the state. A concrete obligation denotes that S is obliged to take A on O Ctx is true before Ctx_v occurs. The obligation is deactivated if the obligation context Ctx ends.

activate Obligation : $Hold_e(S, A,O, start(Ctx))$
initiates $Insert\ Obligation(N, S, A,O,Ctx,Ctx_v)$
if $Obligation(N,R, Act, V,Ctx,Ctx_v)$,
$Empower'(S,R),Consider'(A, Act),Use'(O'\ V)$

deactivate Obligation : $Hold_e(S, A.O, end(Ctx))$
initiates $Remove\ Obligation(N, S, A,O,Ctx,Ctx_v)$
if $Obliged(N, S, A,O,Ctx,Ctx_v)$

7.2 Obligation Fulfillment and Violation

Actions required by concrete obligations are monitored using the event *Holde(S,A,O,obligation Fulfilled)*. This event

indicates that the obliged operation *(S,A,O)* has been performed. The detection of this event triggers an active rule to remove the fulfilled obligation from the state by initiating the action *Fulfill*. Similarly, the detection of the obligation violation context is indicated by the initiation of the action *Violate*.

fulfill_Obligation : $Hold_e$(*S, A,O,obligation_Fulfilled*)
initiates *Fulfill*(*N, S, A,O*)
if *Obliged*(*N, S, A,O,Ctx,*Ctx_v)

violate_Obligation : $Hold_e$(*S, A,O,start*(Ctx_v))
initiates *V iolate*(*N, S, A,O*)
if *Obliged*(*N, S, A,O,Ctx,*Ctx_v)

8. Application Example

In this section, we reconsider the example policy presented in Section 2. Permission *p*1 is presented in Section 4. The remaining permissions may be specified as follows.

Permission(p_2, professors, control, projector, lecture_Application Λ application_Initiator)

Permission(p_3, students, use, white_Board, lecture_Application)

Where the permission contexts *lecture_Application* and *Application_Initiator* are specified as follows:

Hold(S,A,O, lecture_Initiator) ⟵ *Location(S,L),*
Application Initiator(lecture,S,L)

Hold(S,A,O, lecture_Application) ⟵ *Location(S,L),*
Active Application(lecture,L)

To enable the monitoring of the above access control policy, one needs to specify the effects of the occurrence of the domain-dependent actions on the fluents *Application Initiator* and *Active Application*. These effects are specified as follows.

Do(*S, start_Application, App*)
causes *Active_Application*(*App,L*),
Application Initiator(*S, App,L*)
if *Location*(*S,L*)

Do(*S, end Application, App*)
causes ¬ *Active_Application*(*App,L*),
¬ *Application_Initiator*(*S, App,L*)
if *Location*(*S,L*)

Finally, we consider the permission p_4 as an example of authorization policies for collaborative applications. This permission states that "When there is an ongoing meeting in the meeting room, only information to which all present subjects are allowed to consult may be displayed on the screen". This permission may be specified as follows.

Permission(p_4, any_Subject, display, screen_Info,
meeting_Default_Mode)

Where the context *meeting_Default_Mode* is specified as follows:

Hold(S, display, O, meeting_Default_Mode) ⟵
Location(S,meeting_Room), Active_Application(meeting,
meeting_Room), :(Location(S',meeting_Room),
¬ *Permitted(N,S',view,O,Ctx))*

The above context specifies that the context *meeting_Default_Mode* holds for a subject *S* who wants to display an object *O* only if *S* is in a meeting and that there is no subject in the meeting room who is not authorized to view the object *O*.

Several other collaborative contexts may be specified in our framework. For instance, the *collaborate* mode [14] where all subjects who are present share their access rights may be specified using the following context rule.

$$Hold(S, display, O, meeting\_Collaborate\_Mode) \leftarrow Location(S,meeting\_Room), Active\_Application(meeting\_Room, meeting), Location(S',L), Permitted(N,S',view,O,Ctx)$$

9. Related Work

Other access control models have been proposed to enable the specification of contextual access control policies. Some models have focused on particular constraint types. For instance, the GEO-RBAC [4] model considers the restriction of role's visibility to specific geographic areas, and the GTRBAC model [10] introduces mechanisms for the enabling and disabling of roles based on temporal constraints. Our model is more general since it enables the specification of different constraint types including temporal and spatial constraints [6]. We also support obligation policies.

Among models which support general context specification is the GRBAC model [5]. The model introduces environment roles to allow the specification of *environmental* conditions. However, GRBAC does not enable the specifi- cation of personalized contexts since it only captures general state conditions, *e.g.* the time of the day and a high CPU load. The DRBAC model [15] uses state machines to maintain the role subset for a user and the permission subset for each role. The model uses event condition action (ECA) rules to make state transitions. It does not however support permission revocation.

General remarks can be made over the models discussed above. All these models are RBAC-based models. Therefore, they introduce a large number of roles in an access control system which complicates the specification and the management of the policy. In addition, since the central idea to specify contextual conditions in the RBAC model is to specify constraints on the different components of the model namely permissions, roles, and the user-role and permission-role assignments, all these models fail to capture conditions denoting relationships between the permission subject, action and object, *e.g.* consider the permission "a patient's files may be only consulted by the patient's assigned doctor". Therefore, we argue that our model is more suitable for the specification of personalized security rules.

In [11], a programming framework for specifying and enforcing context-based access control requirements is introduced. The presented model introduces several extensions to the RBAC model to enable the specification of personalized permissions and to support ongoing authorizations. In contrast, our model is simpler. Therefore, contextual security rule specification and interpretation are more intuitive and straightforward. In addition, we have presented a formalization of our model. This enables us to use logic-based techniques to analyze and detect inconsistencies in the policy. We also integrate obligation policies.

In [9], a team-based access control using contexts is presented. The model introduces *teams* to represent a group of users having the objective of completing a specific activity in a particular context. Team role permissions may be combined in different ways, *e.g.* using the sum of or the maximum or minimum of access permission sets of team members. The model does not however consider permission monitoring and revocation.

The UCON model [13] is introduced to support a broad range of usage controls. With respect to our work, in UCON, ongoing authorizations are enforced by periodically evaluating decision predicates and access is revoked if the evaluation fails. In contrast, in our framework, the system automatically derives the events which should be monitored to enable an automatic revocation of permissions. The *UCON* model also does not support the specification of general or global obligations since obligations are always associated with resource usage.

In [12], the secure evaluation of authorization policy conditions is studied and a generic condition specification and evaluation service is presented. In contrast, we have studied the contextual management of security policies as opposed to how context can be acquired or evaluated in a secure way. In a practical security framework, both these aspects surely need to be addressed.

In [6], the expression and evaluation of different types of contexts (*e.g.* temporal, spatial, provisional, etc) in the *OrBAC* model is studied. However, context monitoring and dynamic update of the authorization policy are not considered. This work presents an extension to the work in [6] which addresses this issue. Furthermore, we show how this enables the integration of obligations in the policy language.

10. Conclusion

In this paper, we presented a formal contextual security policy model for pervasive environments. The model supports the specification of personalized contextual authorization and obligation policies, dynamic permission revocation, and the specification of policies of collaborative applications. Future work consists of integrating group obligations [7] into the model.

References

[1] Abou El Kalam, A., Baida, R. E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y.Mi'ege, A. Saurel,C., Trouessin,G. (2003). Organization Based Access Control. *In: POLICY*.

[2] Bandara, A.,Craven, S. C. R. , Lobo, J., Lupu, E. , Ma, J., Russo, A., Sloman, M.(2009). Expressive policy analysis with enhanced system dynamicity. *ASIACCS*.

[3] Baral, C., Lobo, J. (1996). Formal characterization of active databases, *In:* Proceedings of the International Workshop on Logic in Databases, p. 175–195, London, UK, 1996. Springer-Verlag.

[4] Bertino, E., Catania, B., Damiani, M. L.,Perlasca, P. (2005). Georbac: a spatially aware rbac. *In: SACMAT*, p. 29–37, New York, NY, USA, ACM.

[5] Covington, M. J., Long, W.,Srinivasan, S., Dev, A. K,. Ahamad, M., Abowd. G. D (2001). Securing context-aware applications using environment roles, *In:* SACMAT, p. 10–20, New York, NY, USA, ACM.

[6] Cuppens, F., Cuppens-Boulahia,N.(2008). Modeling contextual security policies. *Int. J. Inf. Secur.*, 7(4) 285–305.

[7] El Rakaiby, Y., Cuppens, F., Cuppens-Boulahia, N. (2009). Formalization and management of group obligations, *In: POLICY*

[8] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., Chandramouli,R. (2001). Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4 (3) 224–274.

[9] Georgiadis, C. K., Mavridis, I., Pangalos, G., Thomas,R. K.(2001). Flexible team-based access control using contexts, *In: SACMAT*, p. 21–27, New York, NY, USA. ACM.

[10] Joshi, J. B., Bertino, E., Latif, U., Ghafoor, A (2005). A generalized temporal role-based access control model, *IEEE Transactions on Knowledge and Data Engineering*, 17 (1) 4–23.

[11] Kulkarni, D., Tripathi, A. (2008). Context-aware role-based access control in pervasive computing systems. *In:* SACMAT, p. 113–122, New York, NY, USA. ACM.

[12] McDaniel, P. (2003). On context in authorization policy, *In:* SACMAT, p.80–89, New York, NY, USA, ACM.

[13] Park,J., Sandhu, R. (2004). The *uconABC* usage control model, *ACM Trans. Inf. Syst. Secur*, 7.

[14] Sampemane, G., Naldurg, P., Campbell, R. (2002) Access control for active spaces. *In:* ACSAC.

[15] Zhang, G., Parashar,M. (2003). Dynamic context-aware access control for grid applications. *In:* GRID, p. 101–108.