INTERACTIVITY FOR REACTIVE ACCESS CONTROL

Yehia ElRakaiby, Frederic Cuppens and Nora Cuppens-Boulahia
Networks, Security and Multimedia Department, TELECOM Bretagne, 2 rue de la chataigneraie, Rennes, France

Keywords: Access Control, Multi-way Control, Interactivity, Pervasive Environments.

Abstract: Technological advances enhanced the computing and communication capabilities of electronic devices bringing us new pervasive environments where information is present everywhere and can be accessed from anywhere. These environments made way to new intelligent and context-aware applications which have more sophisticated access control requirements. So far, there have been two main categories of access control systems: passive security systems which evaluate access requests according to static predefined permissions; and dynamic security systems which integrate the context in the evaluation of access requests. These models can thus be justly classified as anticipative models since all security rules have to be completely defined before an access request is made. In this paper, we present a formal access control model that extends context-based models to allow just-in-time specification of access control policies. The model relies on interactivity to support active participation of users in the evaluation of the security policy, thus enabling them to participate in the definition of the access policy at the time of the request.

1 INTRODUCTION

Technological advances in computers and networks have enabled new device and networking capabilities bringing us new service models, applications and lifestyles. These pervasive environments require sophisticated access control systems to fulfill the new application and user requirements. In this paper, we present a formal model that extends context-based access control models by allowing the specification of some aspects of the access control policy at the time of the request. So far, there have been two main categories of access control systems: passive security systems which evaluate access requests according to static predefined permissions *e.g.* the $RBAC_0$ model (R.S. et al., 1996); and dynamic security systems which integrate the context in the evaluation of access requests *e.g.* GRBAC (Moyer, 2001) and OrBAC (Cuppens and Miège, 2003). In these systems, access request evaluation relies on the consultation of predefined rules. Therefore, they require that the administrator foresee all possible access requests and configure how each is to be evaluated *in advance*. Thus, these systems can be justly characterized as *anticipative models* since all security rules have to be completely defined before an access request is made.

However, since the future is inherently unpredictable, it is often more effective to create more options for the policy designer. We argue that access control systems can go one step further by allowing the specification of the access control policy at the time of the request through interactivity with subjects. This approach has many benefits: Firstly, policy designers do not have to fully define the access policy and some aspects of the policy may be left unspecified until subsequent access requests are made. Secondly, they permit subjects to be aware of access operations to sensitive objects which can be an important requirement of a security system. Finally, it makes possible to deal with unexpected access requests if it is for any reason impossible to foresee all possible access requests. For example, a security rule may specify that in the case of an emergency, access to a patient's file by one of the hospital's doctors has to be authorized *online* by one of the patient's parents.

In this paper, we present an access control system model that allows policy designers to fully define security policy *in advance* as in traditional models or at the time of the request. This composite approach makes the model capable of supporting requirements that are difficult to fulfill using traditional models. We also show how our policy can be enforced using event-condition-action (ECA) rules.

The article is organized as follows. In section 2, we discuss related work and motivate our approach. Section 3 introduces informally the system architecture and operation. In section 4, the language and the concepts used for the formalization of the model are introduced. Section 5 presents the model's formal representation. In section 6, we discuss the policy interpretation and enforcement. Section 7 shows some use cases. Section 8 briefly discusses a prototype implementation of the model. Finally, section 9 gives our conclusions and discusses future work.

2 RELATED WORK & MOTIVATION

Access control in pervasive environments has attracted many researchers in the last few years. Most works have focused on introducing *RBAC* extensions to add context-awareness and dynamic activation of permissions to the model (Moyer, 2001; Zhang and Parashar, 2004). In accordance with the spirit of the *RBAC* model, these extensions have introduced the notion of contextual or environmental roles which are roles that are activated/deactivated as a result of the occurrence of events in the system. Consequently, permissions given to subjects are updated whenever relevant changes in the system are detected.

In this paper, we tackle a different problem which is the active access control decision-making. Active access control may be also considered as one possible solution to the multi-way control problem. Multi-way control refers to the situation where there are more than one party that hold some rights over a resource and therefore may request to exercise some control over its use. To the best of our knowledge, this issue has been rarely identified (Park and Sandhu, 2004) or dealt with in access control systems. In most systems today, control decision is one-way where the resource provider solely determines the access requester's access rights. A patient file on some hospital's database is a typical example of situations where multi-way decision may be necessary as it may be required that the patient be an *active* party in the access control *online* decision-making process to his/her personal files. Today's pervasive environments offer the necessary mechanisms to support active multi-way control decisions and the benefits of having active access control are clear: awareness of important accesses, just-in-time specification of the access control policy and dealing with some possibly unexpected accesses.

Requiring a subject's consent before authorizing access to his objects has been considered in the work of Becker and Sewell (Becker and Sewell, 2004). They have studied the specification of access control policy in an Electronic Health Record system based on the requirements for the UK health Service procurement exercise. Getting the patient's consent for accessing his/her personal information was one of the requirements. They have proposed to solve the problem using role activation: to allow a clinician to be e.g. a Treating-physician and thus enabling the clinician to access his/her files, a patient has to activate a consent role. In order for the patient to be allowed to activate a consent role, the clinician has to have first asked the activation of the role which is represented by the activation of a consent-request role. In our model, consent is just a subset of the control operations that the patient is allowed to perform as the patient can choose the set of operations to allow such as "my clinician can view only my last medical record and not my entire history" or "can only read but not write" or specify a set of requirements of the access such as the clinician can look at my file but "only at the hospital". The specification of such requirements using a role approach is more complex as it requires the specification of different roles corresponding to the different possible sets of operations and the specification of a set of role activation constraints to model the context in which the permission is to be granted. We believe that our approach is more intuitive as it is more direct and avoids an unnecessary indirection created by the use of roles.

In (Goffee et al., 2004), a decentralized PKI-based authorization for wireless LANs is presented. The authors studied granting access to the wireless network only after they are authorized by local users. They have proposed a PKI-based authorization mechanism allowing internal users to delegate access rights to guests: first, the guest uploads its public key on a web server, the internal user searches for the guest's key and signs it. The guest uses the signed certificate to access the wireless network. Their work shows the relevance of our proposal to distributed network environments as it is an example of situations where access to the resource (the wireless network) needs to be authorized by some subject (the local users) when the access request is made (when the guest uploads its certificate).

Stiemerling and Wulfin (Stiemerling and Wulf, 2004) investigated the necessary access control mechanisms for group interaction. They identified several potential requirements namely: the role of a trusted third person, awareness and negotiation. They have developed and implemented six technical mechanisms to fulfill the aforementioned requirements. What distinguishes our work with respect to theirs is that our access control decision-making is context-aware allowing better expressiveness of access control requirements. Additionally, we have proposed a formal model for our access control system and proposed an appropriate enforcement mechanism of our policy in the form of event-action-condition (ECA) rules. Although our current model does not support negotiation, it is one possible extension to this work (Haidar et al., 2007).

PDL (Lobo et al., 1999) and *Ponder* (Damianou et al., 2001) are two highly-cited policy specification languages. Policies in *PDL* and obligation policies in *PONDER* are expressed in the form of *ECA* rules. Our use of *ECA* rules is different since active rules in our framework are domain independent rules which are used to interpret and enforce the policy. In other words, we separate between the policy expression and the policy interpretation. In *PDL* and *PONDER*, active rules are domain dependent rules that represent the system policy.

3 SYSTEM DESCRIPTION & OPERATION

The proposed system architecture is centered around the access control system which communicates with entities of its environment through messages. The environment includes subjects and context providers. Subjects are *active* entities in the environment capable of interaction with the system. Context providers store contextual information relevant to the evaluation of the access control policy *e.g.* presence servers, localization servers, etc.

The access control system evaluates its policy when messages are received from the system subjects. The policy evaluation process is shown in figure 1: when an access request is received (1), the policy is evaluated (2). We consider two different types of permissions in the policy: *contextual permissions* which are permissions that are associated with a set of conditions that must be true for the permission to be *active* (3); and in which case the access request is *granted* (4). The other type of permissions are *dynamic permissions*. Dynamic permissions require interaction with the subject who manages the resource. Interaction with resource-managers is accomplished using two messages namely the system-request message that is sent to the resource manager (4') and the subject-response message which represents the subject response to the access control system (5'). In the user response, the resource manager can authorize or deny the access, limit the current access to a subset of the operations defined in the dynamic permission or specify some extra conditions that are verified by the system before access is granted (6'). If the newly defined conditions are true, a grant is issued (7'). Consider the following example from an intelligent home environment: suppose Jack wishes to control access to his CD collection. Jack can set up a dynamic-permission rule that states that he is to be contacted when an access request is made to his CD collection by a member of his family. Thus, when an access request is made, Jack is contacted. Jack can subsequently specify several access requirements that dynamically apply on that particular access request: for example, he can limit the access to *only his rock music collection* or if the access requester is one of his sons, Jack can specify some conditions such as his son can access his CD collection but *only if he has done his homework*; or Jack can control what operations are authorized during the access such as *only read operations* are permitted.

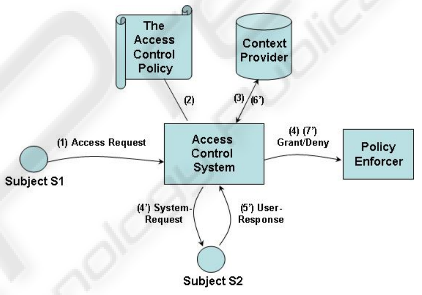

Figure 1: System Architecture.

4 BASIC CONCEPTS

OrBAC Policies. Security policies in the *OrBAC* model (Abou El Kalam et al., 2003) are tied to an organization. An organization represents the authority that has issued the policy and to which the policy pertains; thus the organization entity plays an important role in the administration of security policies (Cuppens and Miège, 2004). In a distributed environment where every subject specifies policies that apply to his managed resources, the different subject policies are easily modelled in the *OrBAC* model by considering each subject as a separate organization. However, since administration is not our primary objective, throughout the paper, we consider one policy pertaining to one single organization; and therefore, all references to the organization entity are omitted in the following.

Policy Element Representation. System objects are subjects, resources and actions. Objects are represented using constant symbols starting with lowercase letters *e.g.* tom, newspaper and read. Variables are represented using capital letters identifiers. Variables and constants are *terms* of the language.

Relations between system objects are described by relation symbols called predicate symbols *e.g.* $Permission(subject, operation)$ is a binary relation of type Permission and is a relation linking the subject and operation. Relation symbols are represented by identifiers starting with capital letters. Facts describe permissions in the system and are expressed using atomic formulas *e.g.* $Permission(tom, O(read, newspaper))$ is a permission for Tom to perform the operation $(read, newspaper)$.

Object attributes are represented using the predicate symbol $Attribute(AttrType, Object, Value)$ which states that the attribute $AttrType$ of the object $Object$ has the value $Value$. The variable symbol $AttrType$ is a variable over the set of attribute types, and $Value$ is a variable over the attributes of objects.

OrBAC Contexts. Contexts are requirements or conditions on object attributes and on the current state. The context modelling and evaluation follows the context modelling of the Organization-based access control model (Cuppens and Miège, 2003). The specification of contexts in *OrBAC* is separated from the permission which allows context reusability, context composition and easier interpretation and specification of policy rules. *OrBAC* contexts are evaluated using the predicate $Hold(S,A,R,Context)$ where $Context$ is an identifier of the set of conditions on the subject S requesting to perform the action A on the resource R. The context definition may also specify conditions on the system's state. In order to specify conjunctive, disjunctive and negative conditions, a simple context language with three operators AND, OR and NOT operators is used:

- $Hold(S,A,R,Ctx_1\&Ctx_2) \leftarrow Hold(S,A,R,Ctx_1), Hold(S,A,R,Ctx_2)$
- $Hold(S,A,R,Ctx_1 \vee Ctx_2) \leftarrow Hold(S,A,R,Ctx_1) \vee Hold(S,A,R,Ctx_2)$
- $Hold(S,A,R,\neg Ctx_1) \leftarrow \neg Hold(S,A,R,Ctx_1)$

For example: the set of requirements that the access requester's age be less than 10, that he/she be at school and that the request be made during the morning can be specified using the following two contexts:
$Hold(S,A,R,childAtSchool) \leftarrow$
$Attribute(age,S,X), X < 10, Attribute(location,S,school)$

$Hold(S,A,R,morning) \leftarrow$
$after\_time(08:00), before\_time(12:00)$

The composed context $Hold(S,A,R,childAtSchool \& morning)$ represents the desired set of requirements.

Organizational Entities. System objects are managed using the organizational entities *roles*, *views* and *activities*. Roles, views and activities are used to group subjects, resources and operations, respectively. Policies defined over organizational entities apply to all its *members* (Sloman and Twidle, 1994). Roles, views and activities are hierarchical and a policy that applies to a role, view or an activity domain propagates to its *sub-roles*, *sub-views* and *sub-activities*, respectively. The used base relations defining roles are shown in table 1. Hierarchy over views and activities is similarly defined. Every resource has a *type*. Every resource type supports a number of actions. A system operation is represented by the function symbol $O(Action, Resource)$. To facilitate the organization of objects and the policy definition, we logically interconnect views and activities using the relations defined in table 2.

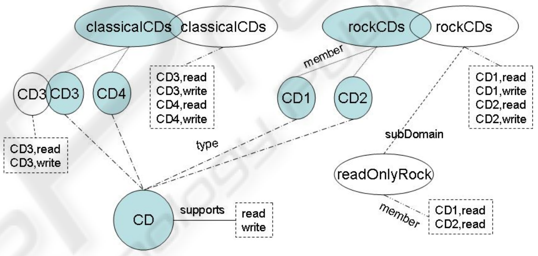

Figure 2: Object Organization Example.

Consider the example shown in figure 2: subject Jack wants to organize his CD collection composed of two rock CDs CD_1 and CD_2; and two classical CDs CD_3 and CD_4; and that a resource type CD supports the operations read and write. Jack creates the two views *rockCDs* and *classicalCDs* and adds the two rock CDs and the two classical CDs to the *rockCDs* and the *classicalCDs* views respectively. The logical consequences of the relations defined in table 2 are: every resource represents an activity containing the operations supported by the resource; for example, CD_1 is an activity containing the operations $\{O(read, cd_1), O(write, cd_1)\}$; and every view is an activity containing all the operations of its derived resource members; thus the activity *rockCDs* contains $\{O(read, cd_1), O(write, cd_1), O(read, cd_2), O(write, cd_2)\}$. To specify a restricted subset of the operations of the activity *rockCDs*, Jack creates a sub-activity *readOnlyRockCDs* from *rockCDs* to which he adds as members the operations $\{O(read, cd_1), O(read, cd_2)\}$.

5 FORMAL MODEL

Formally, the model is represented as follows:

Table 1: Organizational Entities.

Symbol	Description
$Subject(Sub)$	Specifies that *Sub* is a subject of the system
$Role(Role)$	Specifies that *Role* is a role. Roles are considered also subjects of the system. Therefore the following rule is defined: $Subject(Role) \leftarrow Role(Role)$
$RoleMember(Sub, Role)$	Holds if the subject *Sub*, is assigned the role *Role*.
$SubRole(Role1, Role2) \leftarrow Role(Role1), Role(Role2), RoleMember(Role1, Role2), (Role1 \neq Role2), \neg SubRole(Role2, Role1)$	Holds if the role *Role*1 is a sub-role of *Role*2. The body of the rule is used to ensure that there are no cyclic relationships in the role structure.
$RoleDerivedMember(Sub, Role) \leftarrow RoleMember(Sub, Role)$ $DerivedMember(Sub, Role) \leftarrow SubRole(SubR, Role), RoleDerivedMember(Sub, SubR)$	Determines the membership of a role across the entire structure. The first rule identifies all direct members of the role *Role*. The second rule recursively identifies members of sub-roles of the role *Role*.

- The System Basic Elements
  - The sets: Subjects (S), Resources (R), resource-Types (T), Actions (A), Operations (O), Attributes (Att) and Contexts (C).
  - Dynamic Context (C_d) is of type boolean $C_d \in \{true, false\}$. If $true$, C_d represents a requirement to contact the resource manager for access authorization.
- The Organizational Entities
  - Roles ($\mathcal{R}$), Views ($\mathcal{V}$), Activities ($\mathcal{A}$) representing the sets of the system roles, views and activities respectively. Membership of organizational entities is determined using the predicates *Member* and *DerivedMember* of table 1.
- The Policy consists of a set of contextual and dynamic permissions formally represented as follows:
  - $P \subseteq \mathcal{R} \times \mathcal{A} \times C \times C_d$
  - Ex: $P(family, rockCDs, atHome, true)$ is a dynamic permission which represents an active authorization requirement when a member of the role $family$ requests to perform operations of the activity *rocksCDs* if the subject requesting access is at home. The context $atHome$ is thus defined as follows:
    $Hold(S, R, A, atHome) \leftarrow$
    $Attribute(location, S, atHome)$
- The System Messages
  - Access-request (AR): represents a subject's request to perform an activity: $AR \subseteq S \times \mathcal{A}$
  - Grant(GR): represents the acceptance of the access request: $GR \subseteq S \times O$
  - System-Request Messages (SR): represent the possible messages sent by the system to subjects: $SR \subseteq S \times S \times \mathcal{A} \times ID$
    represents that the subject S is contacted to authorize the access request $S \times \mathcal{A}$. ID is a unique interaction identifier.
  - Manager-response Messages (MR): represent the resource-manager response to a system-request: $MR \subseteq S \times \mathcal{A} \times C \times ID$

6 POLICY INTERPRETATION & ENFORCEMENT

The policy is enforced using domain independent event-condition-action (ECA) rules. *ECA* rules have well-defined semantics and avoid having to resort to custom-implementations for the interpretation and enforcement of security policies. The use of *ECA* rules for the enforcement of *RBAC* and its different extensions in a uniform manner has been investigated in (Adaikkalavan and Chakravarthy, 2005). An ECA rule is of the form:
on *event* **if** *condition* **then** *action*
The rule is read as: When *event* occurs and *condition* is true, *action* is executed. *Events* in the system are messages received from subjects namely *access-request* and *manager-response* messages. Possible actions are the output messages *grant-access*, *deny-access* and *system-request* and the data management operations *add* and *delete*. In the case of interaction, the function *create* is used to generate a unique identifier for every system initiated interaction. The system's policy is default-deny; in other words, if a permission is not found for some access request, access is denied. The system behavior is therefore modelled and enforced by the following three active rules:

- The Access-Request/Grant Rule:
  on $AR(S_1, \mathcal{A}_1)$
  if $P(\mathcal{R}_2, \mathcal{A}_2, Context, false)$,
  $DerivedMember(S_1, \mathcal{R}_2)$,
  $Compatible(\mathcal{A}_1, \mathcal{A}_2)$,

Table 2: Resources and Operations.

Symbol	**Description**
$Resource(R)$	States that R is a system resource
$ResourceType(R,T)$	States that the type of R is T
$ResourceManager(S,R)$	States that the subject S is the manager of resource R
$Supports(T,A)$	States that the resource type T supports the action A
$Operation(R,A) \leftarrow Resource(R), ResourceType(R,T), Supports(T,A)$	Represents the different possible operations in the system
$Activity(R) \leftarrow Resource(R)$	Specifies that if R is a system resource identifer then it is also an activity
$ActivityMember(operation(R,A),R) \leftarrow Resource(R), \neg View(R), Operation(R,A)$ $ActivityMember(operation(R,A),Act) \leftarrow View(Act), ResourceDerivedMember(R,Act), \neg View(R), Operation(R,A)$	Holds if the operation $Operation(R,A)$ is a member of the activity Act. When the resource is not a view, the first relation adds the operations supported by the resource to the activity having the same identifier as the resource. If the resource identifier represents a view, the second relation adds the operations supported by all the derived resources from the view, which are not views, to the activity with the same identifier as the view
$Compatible(\mathcal{A}_1,\mathcal{A}_2) \leftarrow SubActivity(\mathcal{A}_1,\mathcal{A}_2) \vee (\mathcal{A}_1 = \mathcal{A}_2)$	The $Compatible$ predicate Holds if $\mathcal{A}_1$ is a sub-activity of or equals to the activity $\mathcal{A}_2$
$Hold(S,\mathcal{A}_1,Context) \leftarrow DerivedMember(Operation(R,A),\mathcal{A}), Hold(S,R,A,Context)$	The $Hold(S,\mathcal{A}_1,Context)$ is true if the defined context Holds for one of the operations that are derived members of the activity.

$DerivedMember(Operation(R,A),\mathcal{A}_1)$,
$Hold(S_1,R,A,Context)$
then $Grant(S_1,Operation(R,A))$
The rule states that when an access request to perform some activity occurs, if the request matches one of the system contextual permissions, a grant is issued to all operations that are derived members of the requested activity and for which the context holds.

- The Access-Request/System-Request Rule:
  on $AR(S_1,\mathcal{A}_1)$
  if $P(\mathcal{R}_2,\mathcal{A}_2,Context,true)$,
  $DerivedMember(S_1,\mathcal{R}_2)$,
  $Compatible(\mathcal{A}_1,\mathcal{A}_2)$,
  $Hold(S_1,\mathcal{A}_1,Context)$,
  $ResourceManager(M,R)$
  then $create(id)$, $add(Interaction(S_1,\mathcal{A}_1,id))$,
  $SR(M,S_1,\mathcal{A}_1,id)$
  The rule states that when an access request to an activity is made and the request matches one of the system's active dynamic permissions, and the context holds for one of the operations specified in the permission, the resource manager is to be contacted to authorize the access. In the present paper, we consider that every resource has exactly one manager:
  $ResourceManager(M_1,R)$,
  $ResourceManager(M_2,R) \rightarrow M_1 = M_2$
  To mark the start of an interaction with a resource manager, a fact $Interaction$ is added to the system's knowledge base.
- The Manager-Response/Grant rule:
  on $MR(S,\mathcal{A},Context,id)$
  if $Interaction(S_1,\mathcal{A}_1,id)$,
  $S = S_1$, $Compatible(\mathcal{A},\mathcal{A}_1)$,
  $DerivedMember(Operation(R,A),\mathcal{A})$,
  $Hold(S,R,A,Context)$
  then $Grant(S,R,A)$, $delete(Interaction(S_1,\mathcal{A}_1,id))$
  The rule states that when a manager-response message is received and if the message corresponds to an ongoing interaction, a grant is issued to all operations that are derived members of the activity specified in the new permission given by the resource manager and for which the context holds. To mark the end of the interaction with the resource manager, the fact $Interaction$ is removed from the system.

6.1 Conflict Detection and Resolution

Several techniques have been proposed for conflict resolution for policy-based systems (Cuppens et al., 2007). In the model, only positive permissions are considered. Thus, the only possible conflict is when a contextual and a dynamic permission are activated at the same time. The conflict is *dynamically* resolved by prioritizing the dynamic permission active rule. Assigning priorities to active rules is supported by almost all active databases and is one way to guarantee the confluence property. Prioritizing dynamic permissions allows us to specify backup contextual permissions when the interaction with the resource manager does not end appropriately as discussed in the next section.

6.2 Handling Timeout

It is important to plan for situations when the manager of the resource does not reply to the system-request message. These situations are managed using simple timers. When a resource-manager is contacted, a

timer is set. If the timer elapses, a default system behavior is executed. For this purpose, our definition of dynamic contexts is modified and two parameters, namely a *deadline* and a *default action*, are added to the definition of dynamic contexts:
$C_d \subseteq D \times DA$
where D is an integer representing the delay after which a timeout occurs and DA represents the system's default action when a timeout is reached and can be any of the following: $DA \in \{accept, deny, other\}$. When $DA = accept$, the access request is accepted when the timeout is reached, when $DA = deny$, the access is denied and when $DA = other$ the system checks if the access request is allowed by one of the policy's contextual permissions. The default setting of the $default$ parameter is $deny$. The above behavior is modelled and enforced by the following active rules:

- The Create-Timer Rule:
  on $add(Interaction(S_1, \mathcal{A}_1, C_d(D, DA), id))$
  then $create(timer(id), D)$
  The rule initiates a timer that produces a timeout event when the timer elapses.
- The Timeout Rule(1):
  on $timeOut(id)$
  if $Interaction(S_1, \mathcal{A}_1, C_d(D, DA), id)$, $DA = deny$
  then $Deny(S_1, \mathcal{A}_1)$
- The Timeout Rule(2):
  on $timeOut(id)$
  if $Interaction(S_1, \mathcal{A}_1, C_d(D, DA), id)$, $DA = accept$, $DerivedMember(opertion(R,A), \mathcal{A}_1)$
  then $Grant(S_1, operaion(R,A))$
- The Timeout Rule(3):
  on $timeOut(id)$
  if $Interaction(S_1, \mathcal{A}_1, C_d(D, DA), id)$,
  $DA = other$, $P(\mathcal{R}_2, \mathcal{A}_2, Context, false)$,
  $DerivedMember(S_1, \mathcal{R}_2)$, $Compatible(\mathcal{A}_2, \mathcal{A}_1)$,
  $DerivedMember(Operation(R,A), \mathcal{A}_2)$,
  $Hold(S_1, R, A, Context)$
  then $Grant(S, operation(R,A))$

7 APPLICATION EXAMPLE

In this section, we reconsider our example from an intelligent home environment. Jack wishes to control his family's access to his CD collection. He defines the following set of permissions:

- P_1: $P(family, classicalCDs, default, false)$
- P_2: $P(family,\ rockCDs,\ jackAvailable,\ dc(60, other))$
  The context $jackAvailable$ is defined as:
  C_1: $Hold(S, R, A, jackAvailable) \leftarrow Attribute(status, jack, available)$
- P_3: $P(family, onlyReadRockCDs, atHome, false)$
  The context $atHome$ is defined as:
  C_2: $Hold(S, R, A, atHome) \leftarrow Attribute(location, S, home)$

The first is a contextual permission giving members of the role $family$ the right to perform operations in the classical CDs activity in the $default$ context which is a context that is always true. The second is a dynamic permission stating that if Jack is available, the access control system should contact him to authorize requests to perform operations on his rock CDs. When an access request to the rock CDs is made,

- $AR(tom, rockCDs)$

The system contacts Jack. Jack using his mobile, PDA or laptop can specify any of the following:

- Limit the authorized operations to a subset of the operations in the dynamic permission
  $MR(tom, readOnlyRockCDs, default, id)$
- Deny the access
  $MR(tom, rockCDs, false, id)$
- Require the verification of some context; for example that his wife Mary is not at home
  $MR(tom, rockCDs,\ maryNotAtHome, id)$
  $Hold(S, R, A, maryNotAtHome) \leftarrow$
  $\neg Attribute(location, mary, atHome)$

If a timeout occurs after the delay specified in the dynamic permission, the system enforces the default action specified in the dynamic permission, in our case, the specified action is *other* thus, the access is checked against the system's contextual permissions and therefore the access is granted only to operations that are members of the activity $readOnlyRockCDs$.

8 IMPLEMENTATION

We have developed the basic functionalities of the model using the rule engine Jess: a java-based rule engine combining declarative logic programming with object-oriented programming. Jess supports both

The prototype uses a mixture of java code (for procedure calls and various interactions with the environment such as the sending and receiving of messages) and jess code (to code the policy rules and logical derivations). The prototype works as follows: the access control system first loads the policy rules coded in jess. Then it loads the permissions from a simulated database into the engine's working memory. When an access request is received, a fact representing the request is injected into the policy engine's working memory. The engine evaluates the request according to the access policy rules. When there are remote conditions to evaluate, information is retrieved from the appropriate context provider. At the end of the processing, the rule engine produces the policy decision reached which is then enforced by the appropriate mechanisms.

9 CONCLUSIONS AND FUTURE WORK

In this article, we have introduced an access control model capable of supporting interactivity with users to enable them to specify aspects of the access control policy at the time of the access request. We have shown how it is possible to express clearly access requirements difficult or impossible to express in traditional models and how different contextual conditions fit into the model.

We are currently working to add support of ongoing controls to the model (Park and Sandhu, 2004). An important security requirement that we are also looking into is to enable the user to add static rules to the access control system *e.g.* to deny for always or for a certain period of time requests from a particular subject as we recognize that the model is vulnerable to some social engineering attacks. We are also considering integrating delegation with interactivity to allow the users to delegate their capabilities to others in a just-in-time manner (Ben Ghorbel-Talbi et al., 2007). The ability to contact several contacts by having more than one resource manager and handling several user-responses for the same access request is also studied.

Finally, we believe the model will prove to be an interesting model for access control in pervasive and collaborative environments and that it lays the foundation to a new generation of access control systems that integrate interactivity with users to add flexibility to traditional access control systems.

ACKNOWLEDGEMENTS

This research is partially sponsored by the Polux project funded by the French “Agence Nationale de la Recherche”. Yehia El Rakaiby’s PhD thesis is funded by a grant from the Britany region.

REFERENCES

Abou El Kalam, A., Baida, R. E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., and Trouessin, G. (2003). Organization Based Access Control. In *Policy’03*.

Adaikkalavan, R. and Chakravarthy, S. (2005). Active authorization rules for enforcing role-based access control and its extensions. *In Proc. of The 21st IEEE International Conference on Data Engineering (ICDE)*.

Becker, M. and Sewell, P. (2004). Cassandra: distributed access control policies with tunable expressiveness. In *POLICY 2004*, pages 159–168.

Ben Ghorbel-Talbi, M., Cuppens, F., Cuppens-Boulahia, N., and Bouhoula, A. (2007). Managing delegation in access control models. *ADCOM 2007*, pages 744–751.

Cuppens, F., Cuppens-Boulahia, N., and Ghorbel, M. B. (2007). High level conflict management strategies in advanced access control models. *Electron. Notes Theor. Comput. Sci.*, 186:3–26.

Cuppens, F. and Miège, A. (2003). Modelling contexts in the or-bac model. In *19th Annual Computer Security Applications Conference (ACSAC ’03)*.

Cuppens, F. and Miège, A. (2004). Administration Model for Or-BAC. In *Computer Systems Science and Engineering (CSSE’04)*.

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In *POLICY ’01*, pages 18–38, London, UK. Springer-Verlag.

Goffee, N., Kim, S., Smith, S., Taylor, P., Zhao, M., and Marchesini, J. (2004). Greenpass: Decentralized, pki-based authorization for wireless lans. *3rd Annual PKI Research and Development Workshop Proceedings*.

Haidar, D. A., Cuppens-Boulahia, N., Cuppens, F., and Debar, H. (2007). Access negotiation within xacml architecture. *SARSSI. Annecy, France.*

Lobo, J., Bhatia, R., and Naqvi, S. (1999). A policy description language. In *Sixteenth national conference on Artificial intelligence*, pages 291–298, Orlando, Florida, United States.

Moyer, M. J. (2001). Generalized role-based access control. In *21st International Conference on Distributed Computing Systems*.

Park, J. and Sandhu, R. (2004). The uconabc usage control model. *ACM Trans. Inf. Syst. Secur*, pages 128–174.

R.S., S., Coyne E.J., F. H., and C.E., Y. (1996). Role-based access control models. *IEEE Computer*.

Sloman, M. and Twidle, K. (1994). Domains: a framework for structuring management policy. pages 433–453.

Stiemerling, O. and Wulf, V. (2004). Beyond ”yes or no” - extending access control in groupware with awareness and negotiation. *Group Decision and Negotiation*, pages 221–235.

Zhang, G. and Parashar, M. (2004). Context-aware dynamic access control for pervasive computing. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*.