From State-based to Event-based Contextual Security Policies

Yehia ElRakaiby, Frédéric Cuppens and Nora Cuppens-Boulahia
TELECOM-Bretagne
35510, Cesson Sévigné, France
Email: {yehia.elrakaiby, frideric.cuppens, nora.cuppens}@telecom-bretagne.eu

Abstract—In this paper, we present a formal contextual security model for pervasive computing applications. The model supports the specification of access and usage controls, enables the monitoring and the revocation of access rights, supports the specification of personalized context permissions and access control policy reconfiguration required to enable collaborative applications. The model is also logic-based. Therefore, it enables the use of formal policy conflict and dynamic system analysis techniques.

I. INTRODUCTION

Traditional access control models like [1] considered static permissions. More recent access control like [2]–[5] models integrated contextual conditions. Both these systems simply provided a yes or no answer to access requests.

The evolution of information systems and new interactive smart environments introduced new challenges and requirements. In particular, in these highly dynamic environments, one must consider different types of usage control [6]. Usage controls typically specify conditions which must hold before, while or after access is made to the resource. Some usage controls may be expressed in the form of obligations. For instance, consider the two obligations “Before a subject is granted access to the research lab, he or she must be properly authenticated” and the obligation “When a patient is admitted to the emergency room, he or she should be examined by one of the doctors within 10 minutes.” These two obligations represent a pre- and post-use control requirements respectively.

Usage controls can also take the form of state conditions which must hold while access takes place. For instance, consider the permission “a nurse can only access a patient’s file while he or she is in the patient’s room”. To correctly enforce this permission, the nurse’s access to the files need to be monitored and the nurse’s access should be revoked whenever the nurse leaves the patient’s room.

To enable the specification of fine-grained contextual security rules, personalized security rule contexts need to be supported. For instance, consider the permission “a patient’s secret files can only be consulted by his or her assigned doctor during the day”. This permission specifies two constraints. The first constraint specifies a particular relationship between the security rule subject and object namely that the permission subject (the doctor) should be the assigned doctor of the object owner (the patient). The second is a constraint on the system state namely that the access should take place by day.

Therefore, we argue that a contextual security model should enable the specification of conditions on both the system state, and on the security rule subject, action and object and the relationships between them.

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.” To support these applications, authorization policies must be dependent on both the state context and on subjects’ access rights.

In this paper, we propose a security policy management framework for smart environments which meets the above requirements. It supports the specification of both authorization and obligation policies. It supports ongoing authorizations, i.e. permissions in the system are monitored and revoked if their associated context no longer holds. The model enables the specification of personalized security rule contexts and supports collaborative applications. Finally, our framework is logic-based therefore it enables the use of policy-based analysis techniques for dynamic systems and for conflict detection and resolution.

This paper is organized as follows. Section 2 presents a motivating example. Section 3 introduces the basic concepts used in the formalization of our framework. Our access control policy language and the dynamic management of the access policy are presented in Section 4 and Section 5 respectively. Section 6 presents our context language. The obligation policy language and management are presented in Section 7. An application example is presented in Section 8. Finally, Section 9 discusses related works and Section 10 concludes the paper.

II. MOTIVATING EXAMPLE

To motivate our work, we consider the following security policy in an intelligent campus. Some of the examples are inspired from [7].

\[ p_1: \text{A professor may start a lecture only when there is more than 5 students present.} \]

\[ p_2: \text{When a professor starts a lecture in the classroom, only he or she is allowed to control the projector. Students in the space may only read and write on the white board.} \]

\[ p_3: \text{During a board meeting, only information that all present subjects are allowed to consult may be displayed on the screen.} \]
When the professor starts a lecture, he or she should turn on the video projector within 5 minutes.

To correctly manage and enforce the above policy, we identify the following requirements. First, we need to enable the specification of both access and obligation policies. Secondly, we need to support dynamic access control policy reconfiguration to handle permission revocation. Thirdly, we need to support personalized contexts, e.g. the meeting application. Fourthly, we need to enable the specification of access control policies for collaborative applications, e.g. the meeting application in $p_3$. Finally, we add the requirement to formalize policy specification and management to enable the development of the logic-based policy analysis techniques for dynamic systems [8], [9].

III. BASIC CONCEPTS
A. 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 $N$, predicate symbols $Q$ and variables $V$. To enable the specification of security rules for groups of subjects, actions and objects, we also consider the sets roles ($R$), activities ($A$) and views ($V$), 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. $A$ is the rule head and $B_1,...,B_n$ is the rule body. 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 are called fluents to denote that they can change over time.

B. The OrBAC model

The Organization-Based Access Control Model is a logic-based model which enables the specification of contextual security policies. In this paper, the following basic OrBAC relations are introduced:\footnote{In our framework, security policies are sets of permissions, prohibitions, obligations and dispensations. However, in this paper, only permissions and obligations are considered.}

- **Empower** is a predicate over domains $S \times R$. 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 $O \times V$. 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 $A \times A$. If $a$ is an action and $a'$ is an activity, then $Consider(a, a')$ means that action $a$ implements activity $a'$.
- **Permission** is a predicate over the domains $\mathcal{N} \times R \times S \times A \times O \times V \times C$ where $SR = R \cup S$, $AA = A \cup A$ and $OV = V \cup O$. In other words, 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 $\mathcal{N} \times S \times A \times O \times C$. Therefore, a relation $Permitted(n, s, a, o, ctx)$ denotes that permission $n$ currently allows the subject $s$ is permitted to perform action $a$ on object $o$ until the context $ctx$ ends. Therefore, a fact $Permitted$ represents a concrete active permission.

- **Hold** Security policies specify whether a subject $S$ is permitted, prohibited, obliged or dispensed to take action $A$ on object $O$. To enable the specification of conditions over the security rule’s triple $(S, A, O)$ and on the system state, the predicate $Hold(S, A, O, Ctx)$ is introduced. The predicate is defined over domains $S \times A \times O \times C$. A relation $Hold(s, a, o, c)$ means that the set of conditions specified by the context $c$ holds for subject $s$, action $a$ and object $o$. The specification of contextual conditions using the $Hold$ predicate is discussed in Section IV. The obligation language is presented in Section VII-A.

C. Dynamic System Modeling ($L_{active}$)

To formalize access and usage control policies management operations, we use the language $L_{active}$ introduced to provide a formal characterization of active databases. The language enables the description of dynamic systems and borrows its concepts from action specification languages. A translation of the language into logical programs is presented in [10].

The alphabet of $L_{active}$ consists of four sorts: (1) Fluents: are time-varying propositions or facts 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 some policy management operations are required. (4) Rule Names: unique identifiers of Event Condition Action (ECA) rules. ECA rules, also called active rules, are used to initiate policy management operations after the detection of events. An ECA rule states that when event occurs and if conditions are true, then actions are executed.

The semantics of $L_{active}$ is given by the following three propositions:

$$(EL)\; a(X) \; causes \; f(Y) \; if \; p_1(X_1) \; \ldots \; p_i(X_i)$$

$$(ED)\; e(Y) \; after \; a(X) \; if \; p_1(X_1) \; \ldots \; p_i(X_i)$$

$$(AR)\; r(X_1) \; when \; e(X) \; initiates \; [a] \; if \; p_1(X_1) \; \ldots \; p_i(X_i)$$

Where the symbols $fp_1,\ldots,p_i$ are fluent symbols, $a$ is an action symbol, $e_1,\ldots,e_i$ 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 $p_1(X_1)\ldots,p_i(X_i)$ are true causes $f(Y)$ to be true in the next state. An event definition proposition (ED) states that if the conditions $p_1(X_1)\ldots,p_i(X_i)$ are true in the state following the execution of the action $a(X)$, then event $e(Y)$ is produced. An active rule proposition (AR) states that every new detection of the event $e(X)$ initiates the execution of the sequence of actions $[a]$ if the rule conditions are true.

The operational semantics of $L_{active}$ 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. One of the main advantages of the triggered rules is that, due to its simplicity, it is amenable to efficient implementation.

IV. ACCESS CONTROL POLICY SPECIFICATION

A. Access Control Policy Specification using OrBAC

In our framework, access control requirements are specified using the OrBAC policy language. For instance, the permission $p_1$ in Section II may be specified as follows:

$$\textbf{Permission}(p_1, \text{professors}, \text{start}, \text{lecture}, \text{more Than } 5, \text{Students})$$

Where the context $\text{more Than } 5, \text{Students}$ may be specified as follows:

$$\text{Hold}(S, \text{more Than } 5, \text{Students}) \Leftarrow \text{Location}(S, \text{classroom}), \text{Nb Students In Classroom}(N, \text{classroom}), N > 5$$

The context definition above specifies that the context $\text{more Than } 5, \text{Students}$ should hold for a subject if he or she is in a classroom and there is more than 5 students in the classroom. Since this context specifies conditions on the state fluents, we call it a state context.

V. DYNAMIC ACCESS POLICY MANAGEMENT

The contextual security policy in Section IV specifies contextual permissions that enables the system to provide a yes or no answer to access requests. However, it does not enable the monitoring and revocation of permissions. In this section, we show how the model may be extended to support permission monitoring and revocation.

Our proposal consists of the following. First, we describe the system dynamic behavior by specifying how action occurrences in the system update the state fluents using $L_{\text{active}}$ effect laws. Using this description and permission state context definitions, we derive two sets of state conditions in the form of $L_{\text{active}}$ event definitions. The first set of events have the form $\text{Hold}(S,A,O,\text{start}(Ctx))$ and specify the state conditions at which the state context $Ctx$ should begin to hold for the subject $S$, action $A$ and object $O$. The events $\text{Hold}(S,A,O,\text{end}(Ctx))$, on the other hand, specify the state conditions at which the state context $Ctx$ should seize to hold. These events are then used to update the applied access policy using active rules.

A. Dynamic State Description

In our framework, we assume that the state evolves, i.e. the state fluents are updated, by action occurrences in the system. Therefore, the system dynamic behavior is specified using $L_{\text{active}}$ effect laws presented in Section III-C.

For instance, the effects of the actions $\text{enter}$ and $\text{exit}$ on the fluents $\text{Location}$ and $\text{Nb Of Students}$ are specified as follows.

$$\text{Do}(S, \text{enter, classroom})$$

causes $\neg \text{Nb Students In Classroom}(N, \text{classroom})$, $\text{Nb Students In Classroom}(N+1, \text{classroom})$ if $\text{Empower}(S, \text{students}), \text{Nb Students In Classroom}(N, \text{classroom})$

$$\text{Do}(S, \text{exit, classroom})$$

causes $\neg \text{Nb Students In Classroom}(N, \text{classroom}), \text{Nb Students In Classroom}(N-1, \text{classroom})$ if $\text{Empower}(S, \text{students}), \text{Nb Students In Classroom}(N, \text{classroom})$

$$\text{Do}(S, \text{enter, classroom})$$

causes $\text{Location}(S, \text{classroom})$

$$\text{Do}(S, \text{exit, classroom})$$

causes $\neg \text{Location}(S, \text{classroom})$

if $\text{Location}(S, \text{classroom})$

B. Deriving State Context Start and End Conditions

From the dynamic state description and state context definitions, we are able to derive using algorithm 1 the state 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 "$A \text{ causes } F$ if $p_1, ..., p_n$". An action $A$ is said to terminate a fluent $F$ if there exists an effect law "$A \text{ causes } \neg F$ if $p_1, ..., p_n$".

Definition 2: A fluent $F$ is said to be positively defined in the context definition of $Ctx$ if $F$ positively appears in the conditions of the state-based context definition of $Ctx$, i.e. "$\text{Hold}(S, A, O, Ctx) \Leftarrow ..., F, ...$". A fluent $F$ is said to be negatively defined in the context definition of $Ctx$ if $\neg F$ appears in the conditions of the state-based context definition of $Ctx$, i.e. "$\text{Hold}(S, A, O, Ctx) \Leftarrow ..., \neg F, ...$".

Definition 3: An action $A$ is said to be an initiator of a 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 context definition of $Ctx$.

Similarly, an action $A$ is said to be a terminator of a context $Ctx$ if $A$ is an 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 context definition of $Ctx$.

Algorithm 1\footnote{To simplify the notation, we write $A$ to denote an action $\text{Do}(S,A,O)$, $Ctx$ to denote a context $\text{Hold}(S,A,O,Ctx)$ and we write $F$ to denote a state fluent usage}, given state-based context definitions and effect laws, returns the event definitions necessary to monitor the activation and deactivation of state-based 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 $\text{start}(Ctx)$ should be detected after the occurrence of $A$ if the $Ctx$ does not hold in the state and its associated conditions are true. If $A$ is, on the other hand, a terminator of the context $Ctx$, the algorithm returns an event definition which states that...
the event \( \text{end}(Ctx) \) should be detected after the occurrence of \( A \) if the \( Ctx \) holds in the state and its associated conditions are no longer true.

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

\[
\text{Hold}_s(S,\ldots,\text{start}(\text{more},\text{Than},5,\text{Students}))
\]
\[
\text{after} \ Do(S',\text{enter},\text{classroom})
\]
\[
\text{if} \ \neg \text{Hold}_s(S,\ldots,\text{more},\text{Than},5,\text{Students}), \Location(S,\text{classroom}), \Nb_{Of}\text{Students}_{In}\Classroom(N,\text{classroom}), N > 5
\]
\[
\text{Hold}_s(S,\ldots,\text{end}(\text{more},\text{Than},5,\text{Students}))
\]
\[
\text{after} \ Do(S',\text{exit},\text{classroom})
\]
\[
\text{if} \ \text{Hold}_s(S,\ldots,\text{more},\text{Than},5,\text{Students}), \neg \Location(S,\text{classroom}), \Nb_{Of}\text{Students}_{In}\Classroom(N,\text{classroom}), N > 5
\]

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

C. State Context Dynamic Management

When the event \( \text{Hold}_s(S,A,O,\text{start}(Ctx)) \) is detected, the fluent \( \text{Hold}(S,A,O,Ctx) \) is inserted into the state to denote that the context \( Ctx \) holds for the triple \( (S,A,O) \) using the active rule \( \text{context Activation} \) below. An active rule is similarly specified to remove the fluent \( \text{Hold}(S,A,O,Ctx) \) from the state when the end of the state context is detected.

\[
\text{context Activation} : \text{Hold}_s(S,A,O,\text{start}(Ctx))
\]
\[
\text{initializes Activate_Context}(S,A,O,Ctx)
\]
\[
\text{Activate_Context}(S,A,O,Ctx)
\]
\[
\text{causes} \ \text{Hold}_s(S,A,O,Ctx)
\]

D. Permission Activation and Deactivation

When permission state contexts are activated or deactivated, the applied access control policy is updated using the following two active rules:

\[
\text{activate_Permission} : \text{Hold}_s(S,A,O,\text{start}(Ctx))
\]
\[
\text{initializes Insert_Permitted}(N,S,A,O,Ctx)
\]
\[
\text{if} \ \text{Permission}(N,R,\text{Act},V,Ctx)
\]
\[
\text{Empower}'(S,SR) \ \text{Consider}'(A,AA), \text{Use}'(O,OV)
\]
\[
\text{deactivate_Permission} : \text{Hold}_s(S,A,O,\text{end}(Ctx))
\]
\[
\text{initializes Remove_Permitted}(N,S,A,O,Ctx)
\]
\[
\text{if} \ \text{Permission}(N,S,A,O,Ctx)
\]

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

\[
\text{Empower}'(S,SR) \leftrightarrow \text{Role}(SR), \text{Empower}(S,R)
\]
\[
\text{Empower}'(S,S) \leftrightarrow \text{Subject}(S)
\]

VI. CONTEXT LANGUAGE

Sometimes, we may need to compose contexts in security rules using the logical operators conjunction (\( \land \)), disjunction (\( \lor \)) and negation (\( \neg \)) to enable the specification of more sophisticated contexts, e.g. the contexts (\( \text{more},\text{Than},5,\text{Students} \lor \text{campus},\text{Director},\text{Present} \)) and (\( \text{low},\text{Noise},\text{Level} \land \text{low},\text{CPU},\text{Load} \)).

To enable the detection of the activation and deactivation of those composed contexts, the following context detection rules are specified. These rules are applied, after an action occurrence, for the finite set of composed contexts that are used in the policy. This assures that composed event detection after action occurrences always terminates.

a) The conjunction (\( \land \)): The start of a conjunction \( (Ctx_1 \land Ctx_2) \) is detected if the start of one of the contexts in the conjunction is detected while the other context holds in the state as specified below. The detection conditions for the end of contexts composed using conjunctions are obtained by replacing the start function term by the end function term.

\[
\text{start}(Ctxt_1 \land Ctxt_2) \leftrightarrow (Ctxt_1, \text{start}(Ctxt_1)) \lor (Ctxt_2, \text{start}(Ctxt_2))
\]

To use the same rules for the detection of state and event contexts, the function \( \text{start} \) above is defined as an idempotent function for basic user-defined event contexts, i.e. \( \text{start}(C_E) \leftrightarrow C_E \).

b) The disjunction (\( \lor \)): The start of a disjunction \( (Ctx_1 \lor Ctxt_2) \) is detected if the start of one of the contexts in the disjunction is detected while the other context does not hold as specified below. The detection conditions for the end of contexts composed using disjunctions are obtained by replacing the start function term by the end function term.
start(Ctx_1 ∨ Ctx_2) ↔
(¬Ctx_2, start(Ctx_1)) ∨ (¬Ctx_1, start(Ctx_2))

c) Negation (¬): The detection conditions of the start and end of negated contexts are simply the inverse of the detection conditions of the original context. To enable the detection of negated contexts at the system initialization, the action Init is used. In our framework, the action Init is used to initialize the system. Therefore, after this action, we generate the instantaneous event Init_Event and start Hold(¬Ctx) for contexts which do not hold at this state. Note that, consequently, at any state following the system initialization, either the fluent Hold(Ctx) or the fluent Hold(¬Ctx) holds for all basic contexts and composed contexts specified in the policy.

start(¬Ctx) ↔ end(Ctx) ∨ (Init_Event & ¬Ctx)
end(¬Ctx) ↔ start(Ctx)

In our framework, the specification of temporal contexts using event contexts is supported using the special action Clock. This action has the attributes: Time, Day, Week, Month and Year representing the different calendars available in the system and enable the specification of absolute and periodic temporal conditions. Relative temporal deadlines for obligation policies may be specified using the context delay(Nb.TimeUnit) which is a context that is detected Nb time units after the instantiation of the security rule. Interested readers are referred to [11], [13] for further details.

VII. OBLIGATION POLICY SPECIFICATION

A. Obligation Policy Language

In our model, 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 [11]. For instance, the obligation o1 in Section II may be specified as follows:

Obligation(o1, professors, turn_On, projector, lecture_Initiated_By Professor, delay(5.minutes))

Where the context lecture_Initiated_By Professor may be specified as follows:

Hold(S, _start(lecture_Initiated_By Professor))
after Do(S, start, lecture)
if Empower(S, professors)

Hold(S, _end(lecture_Initiated_By Professor))

The above obligation species that when a professor starts a lecture, he or she should turn on the projector without 5 minutes. However, if the lecture is ended before its deadline, the obligation is simply deactivated.

B. Obligation Activation and Deactivation

Obligations are managed similarly to permissions: When the obligation context is started, a concrete obligation Obligated(N.S.A.O,Ctx,Ctx_v) is added to the state to denote that subject S is obliged to take action A on object O while the context Ctx is true before the violation context Ctx_v occurs. If the concrete obligation context ends while the obligation holds, the obligation is deactivated and removed from the state.

activate_Obligation : Hold(S, A, O, start(Ctx))
if Obligation(N, R, Act, V, Ctx, Ctx_v),
Empower(S, R, Consider(V, Act), Use(O, V)
deactivate_Obligation : Hold(S, A, O, end(Ctx))
if Obligated(N, S, A, O, Ctx, Ctx_v)

C. Obligation Fulfillment and Violation

In the framework, actions required by concrete obligations are monitored using the event obligation_Fulfilled which is produced whenever a concrete obligation is fulfilled. The detection of this event triggers an active rule that removes 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(S, A, O, obligation_Fulfilled)
if Obligated(N, S, A, O, Ctx, Ctx_v)

violate_Obligation : Hold(S, A, O, start(Ctx_v))
if Obligated(N, S, A, O, Ctx, Ctx_v)

VIII. APPLICATION EXAMPLE

In this section, we reconsider the example policy presented in Section II. Permission p1 has been specified and discussed in Section IV. Permissions p2 and p3 may be specified as follows.

Permission(professors, control, projector, lecture_Application ∧ application_Initiator)

Permission(students, use, white.Board, lecture_Application)

Where the permission contexts above may be 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.S.L)
To enable the monitoring of the above access control policy, the effects of action occurrences on the fluents Application_Initiator and Active_Application are specified as follows. From these effect laws, the events defining the start and end conditions of the state contexts are derived.

\[
\text{Do}(S, \text{start} \text{Application}, \text{App}) \Rightarrow \text{Active} \text{Application}(\text{App}, L), \text{Application_Initiator}(S, \text{App}, L) \iff \text{Location}(S, L)
\]

\[
\text{Do}(S, \text{end} \text{Application}, \text{App}) \Rightarrow \neg \text{Active} \text{Application}(\text{App}, L), \neg \text{Application_Initiator}(S, \text{App}, L) \iff \text{Location}(S, L)
\]

Finally, we consider the permission \( p_4 \) as an example for collaborative applications authorization policies. This permission states that “In a regular board meeting, only information to which all present subjects are allowed to consult may be displayed on the screen”. It may be specified as follows.

\[
\text{Permission(} \text{any Subject, display, screen Info, meeting_Default_Mode})
\]

Where the context meeting_Default_Mode may be specified as follows:

\[
\text{Hold}(S, \text{display}, O, \text{meeting_Default_Mode}) \iff \text{Location}(S, \text{meeting_Room}), \text{Active_Application}(\text{meeting_Room}, \text{meeting}), \neg \text{(Location}(S', \text{meeting_Room}), \neg \text{Permitted}(\text{N}, S', \text{view}, O, \text{Ctx}))
\]

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 allowed to view the object \( O \).

Several other collaborative contexts may be specified in our framework. For instance, the collaborate mode [7] where all subjects may share their permissions may be specified as follows:

\[
\text{Hold}(S, \text{display}, O, \text{meeting_Collaborate_Mode}) \iff \text{Location}(S, \text{meeting_Room}), \text{Active_Application}(\text{meeting_Room}, \text{meeting}), \text{Location}(S', L), \text{Permitted}(\text{N}, S', \text{view}, O, \text{Ctx})
\]

IX. RELATED WORK

Several access control models have been proposed to enable the specification of contextual access policies. Some models focus on the specification of particular constraint types. For instance, the GEO-RBAC [5] model enables to limit a role’s visibility to specific geographic areas, and the GTRBAC model [3] introduces mechanisms for enabling and disabling of roles based on temporal constraints. Our model is more general since it enables the specification of different types of constraints including temporal and spatial constraints [12] and supports the specification of obligation policies.

Among models which support general context specification is the GRBAC model [14]. The model introduces environment roles to capture conditions that are relevant to access control. However, GRBAC does enable the specification 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 [4] 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 parts of the model namely permissions, roles, or assignments, all these models fail to capture conditions denoting relationships between the permission elements e.g. “patient’s files may be only consulted by his or her assigned doctor”. Therefore, we argue that our model is more suitable for the specification of personalized security rules.

In [15], 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 particular, the model introduces the notions of role admission and validation constraints, dynamic object binding, personalized role permissions, context-based permission activation constraints and context-based resource access constraints. In contrast, our model is simpler and, therefore, contextual security rule specification and interpretation are more intuitive and straightforward. In addition, we have presented a formalization of our model which enables the use of logic-based policy analysis techniques. Furthermore, the proposed model does not support obligations.

In [16], 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 [6] 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 in the system at which permissions should be revoked. The UCON model also does not support the specification of general or global obligations since obligations are always associated with resource usage.

In [17], the security needs of policy condition evaluation for authorization policies are studied and a generic framework implementing condition specification, implementation 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 [12], the expression and evaluation of different types of contexts (e.g., temporal, spatial, provisional, etc) in the OrBAC model is studied. However, the issue of context monitoring for the dynamic update of authorization policies is not considered. This work proposes an extension to the work in [12] which addresses this issue.

X. CONCLUSION

In this paper, we have presented a formal contextual security policy model for pervasive environments. The model supports the specification of authorization and obligation policies, dynamic permission revocation, the expression of personalized contexts and the specification of authorization policies for collaborative applications. The main contribution of this paper is that we have showed how contextual logic-based security policy models may be extended to support dynamic policy reconfiguration and ongoing authorizations by simply providing a description of the dynamic system behavior, and then, deriving from state context definitions event definitions specifying state conditions at which the applied policy needs to be updated. Future work consists of integrating group obligations [11] to the model.

REFERENCES
