A Context-Aware Mandatory Access Control Model for Multilevel Security Environments

Jafar Haadi Jafarian, Morteza Amini, and Rasool Jalili

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
{jafarian@ce,m_amini@ce,jalili@}sharif.edu

Abstract. Mandatory access control models have traditionally been employed as a robust security mechanism in multilevel security environments like military domains. In traditional mandatory models, the security classes associated with entities are context-insensitive. However, context-sensitivity of security classes may be required in some environments. Moreover, as computing technology becomes more pervasive, flexible access control mechanisms are needed. Unlike traditional approaches for access control, such access decisions depend on the combination of the required credentials of users and the context of the system. Incorporating context-awareness into mandatory access control models results in a model appropriate for handling such context-aware policies and context-sensitive class association mostly needed in multilevel security environments. In this paper, we introduce a context-aware mandatory access control model (CAMAC) capable of dynamic adaptation of access control policies to the context, and handling context-sensitive class association, in addition to preservation of confidentiality and integrity. One of the most significant characteristics of the model is its high expressiveness which allows us to express various mandatory access control models such as Bell-LaPadula, Biba, Dion, and Chinese Wall with it.

Keywords: Mandatory Access Control, Context-Awareness, Confidentiality, Integrity.

1 Introduction

As computing technology becomes more pervasive and mobile services are deployed, applications will need flexible access control mechanisms. Unlike traditional approaches for access control, access decisions for these applications will depend on the combination of the required credentials of users and the context and state of the system.

Unlike discretionary and role-based access control, mandatory access control models directly address multilevel security environments where information is classified based on its sensitivity; although they have been deployed in commercial sectors too.

Numerous context-aware access control models are presented in literature. Meanwhile, none of these models directly target new security requirements of multilevel environments; while some of them are applicable to such environments with considerable effort. Since mandatory access control has traditionally been used in these environments, a context-aware mandatory access control model seems the most appropriate choice in this regard.

In traditional mandatory access control models, except for some special cases, the security classes associated with entities are usually insensitive to context. However, in some systems, we may need context-sensitive association of security classes. For instance, in most intelligence agencies, the security level of documents decreases by the elapse of time. Moreover, as computing technology becomes more pervasive, applications in multilevel security domains need more flexible mandatory access control policies. Incorporating context-awareness into mandatory access control models gives rise to a flexible and expressive model suitable for management of such context-aware policies and dynamic class associations.

In this paper, we introduce CAMAC as a context-aware mandatory access control model capable of dynamic adaptation of policies with the context and handling context-sensitive class association, in addition to preserving confidentiality and integrity. In fact, CAMAC uses Bell-LaPadula and Biba properties to preserve confidentiality and integrity of information.

The rest of the paper is organized as follows. Section 2 introduces a brief survey on context-aware access control models. In Section 3, CAMAC model is formally described. In section 4, the expressiveness of CAMAC model is scrutinized. In section 5, evaluation of the model is introduced followed by our conclusion.

2 Related Work

Various mandatory access control models and policies have been introduced in literature. Bell-LaPadula [1, 2], Biba [3], Dion [4] and Chinese Wall [5] are examples of such models and policies. Bell-LaPadula and Biba constitute the infrastructure of the CAMAC model, although the definition used here is mostly based on a minimalist approach introduced by Sandhu in [6, 7].

Many researches are targeted to applying context-awareness to the RBAC model. Kumar et al. [8] proposed a context-sensitive RBAC model that enables traditional RBAC to enforce more complicated security policies dependent on the context of an attempted operation. Al-kahtani et al. [9] proposed the RB-RBAC model, performing role assignment dynamically based on users' attributes or other constraints on roles. GRBAC, Generalized RBAC, [10] incorporates three types of roles; subject roles corresponds to the traditional RBAC roles, object roles which are used to categorize objects, and environment roles to capture environmental or contextual information. Context-aware access control is achieved by employment of these role types in specification of access control policies. Zhang et al. [11] proposed DRBAC, a dynamic context-aware access control for pervasive applications. In DRBAC, there is a role state machine for each user and a permission state machine for each role. Changes in context trigger transitions in the state machines. Therefore, user's role and role's permission are determined according to the context. Georgiadis et al. [12] present a team-based access control model that is aware of contextual information associated with activities in applications. Hu et al. [13] developed a context-aware access control model for distributed healthcare applications. The model defines the notion of context type and context constraint to provide context-aware access control.

Ray et al. [14] proposed a location-based mandatory access control model by extending Bell-LaPadula model with the notion of location. In particular, every location is associated with a confidentiality level and Bell-LaPadula no read-up and no write-down properties are extended by taking confidentiality levels of locations into consideration. Based on Baldauf et al.'s classification of context-aware systems [15], location-based mandatory access control model can be categorized as a location-aware system.

3 CAMAC: A Context-Aware Mandatory Access Control Model

Through an example application enabled by a pervasive computing infrastructure in a smart building of a military environment, we discuss motivation for access control models such as ours. The building has many rooms including administration offices, campuses, etc. Sensors in the building can capture, process and store a variety of information about the building, the users, and their activities. Pervasive applications in such an environment allow military forces to access resources/information from any locations at anytime using mobile devices (PDAs) and wireless networks. While classification is still the basis for all the access control decisions, users' context information and application state should also be considered. For example, an officer can only control the audio/video equipment in a conference room if she/he is scheduled to present in that room at that time by the manager in charge. In such applications, privileges assigned to the user will change as context changes. The example above embodies many of the key ideas of the research presented in this paper. To maintain system security for such a pervasive application, we have to dynamically adapt access permissions granted to users as context information changes. Context information here includes environment of the user such as location and time that the user access the resource and system information such as CPU usage and network bandwidth. The traditional mandatory models do not directly address the requirements of such an application and although many context-aware access controls have been proposed in literature, they are not appropriate for environments where security is directly contingent upon classification. This paper aims at presenting a flexible and expressive model appropriate for multilevel security environments where classification of information is an integral property of the environment.

CAMAC is a context-aware mandatory access control model which utilizes contextual information to enhance expressiveness and flexibility of traditional mandatory access control models. Incorporation of context-awareness into the model changes traditional models in two separate ways. Firstly, contextual information can be used to define more sophisticated access control policies. As an example, an access control policy might require that for a subject to acquire a read access to an object, some timing restrictions must be satisfied. CAMAC model allows definition of such sophisticated access control policies. Secondly, the confidentiality and integrity level of entities can change based on contextual information. In traditional mandatory models, the levels initially assigned to entities are not allowed to change based on the circumstances. For instance, confidentiality level of objects might decrease as their lifetime increases (and so become accessible to less trustworthy subjects). CAMAC also allows such dynamic level association based on contextual information.

3.1 Formal Definition of CAMAC

CAMAC model can be formally described as a ten-tuple:

$\langle$*EntitySet, RepOf, ConfLvl, IntegLvl,* λ, ω, *ContextPredicateSet, ContextSet, OperationSet*$\rangle$

in which:

- *EntitySet* is the set of all entities in the system and is composed of four sets: *User, Subject*, *Object* and *Environment*. *User*, *Subject* and *Object* are the set of all users, subjects and objects in the system respectively. *Environment* set has only one member called *environment*.
- *RepOf: Subject* $\rightarrow$ *User* assigns to each subject the user who has initially initiated or activated it. In other words, for $s \in$ *Subject, RepOf(s)* represents the user on behalf of whom the subject s acts.
- *ConfLvl* is a finite ordered set of confidentiality levels[1] such as $\langle c_n, c_{n-1}, \ldots, c_1\rangle$ in which c_n and c_1 are the highest and lowest levels respectively. As in Bell-LaPadula model, each user, subject and object is associated with a confidentiality level. It must be noted that there exist a difference between Bell-LaPadula and CAMAC in terms of confidentiality level. While Bell-LaPadula confidentiality levels are defined by two components (a classification and a set of categories), CAMAC confidentiality levels only include the first component, i.e. classification. In section 5 we show that the second component, set of categories, is contextual information and can be easily incorporated to the model as a context type.
- *IntegLvl* is a finite ordered set of integrity levels such as $\langle i_n, i_{n-1}, \ldots, i_1\rangle$ in which i_n and i_1 are the highest and lowest levels respectively. As in the Biba model, each user, subject and object is associated with an integrity level. Moreover, the above difference also applies here; i.e. CAMAC integrity levels are defined by only a classification component.
- λ is a mapping function which associates each user, subject and object with a confidentiality level: λ*:User* $\cup$ *Subject* $\cup$ *Object* $\rightarrow$ *ConfLvl*
- ω is a mapping function which associates each user, subject and object with an integrity level: ω*: User* $\cup$ *Subject* $\cup$ *Object* $\rightarrow$ *IntegLvl.*
- *ContextPredicateSet* is the set of current *Context predicates* in the system. Each context predicate is a statement about the value of a contextual attribute. More on context predicates will come in section 3.2.
- *ContextSet* is an ordered set of *context types.* A context type is a property related to every entity or a subset of existing entities in the system. A context type $ct \in$ *ContextSet* can be formally described a 5-tuple:
  $ct = \langle ValueSet_{ct}, OperatorDefinerSet_{ct}, RelatorSet_{ct}, EntityTypeSet_{ct}, LURSet_{ct}\rangle$
  More details on *context types* are given in section 3.3.
- *OperationSet* is the set of all operations in the system. An operation $OPR \in$ *OperationSet* can be formally defined as a pair: $OPR = \langle AccessMode_{OPR}, Constraint_{OPR}\rangle$

  More details on *OperationSet* are given in section 3.5.

3.2 Context Predicate

Each *context predicate* is a predicate which represents the value for a contextual attribute. We define a context predicate $cp \in$ *ContextPredicateSet* as a 4-tuple:

[1] Since Biba uses the term 'integrity level', for Bell-LaPadula, we prefer to use the term 'confidentiality level' instead of 'security level'.

$cp = \langle en, ct, r, v\rangle$

where $en \in \{User, Subject, Object, Environment, ValueSet_{ct_1}, \ldots , ValueSet_{ct_n}\}$, $ct \in ContextSet$, $r \in RelatorSet_{ct}$,$v \in ValueSet_{ct}$, and $ct_1,\ldots,ct_n \in ContextSet$. For example, *⟨John, Location, Is, Classroom⟩* is a context predicate and indicates the current location of subject *John*.

Management and updating context predicates is the responsibility of *Context Management Unit (CMU).* The details on the implementation of CMU are beyond the scope of this paper, and will be explained in another paper. Context Managing Framework [16], the SOCAM project [17], CASS project [18], CoBrA architecture [19], the Context Toolkit [20] can be used as an infrastructure in implementation of *CMU*. In general, we assume that CMU updates *ContextPredicateSet* based on changes of environment, users and system and therefore the consistency and accuracy of *ContextPredicateSet* is permanently preserved.

If $\langle E, X, R, V\rangle$ is a context predicate, $X[E][R]$ will indicate the value assigned to entity E for context type X and relator R. In other words, $X[E][R] = V$. For instance if *⟨John, Location, Is, Classroom⟩* ∈ *ContextPredicateSet*, then *Location[John][Is] = Classroom.* If such a context predicate does not exists in *ContextPredicateSet*, we will assume that $X[E][R] = \perp$ (read as null).

3.3 Context Type

Informally, *a context* is a property related to every entity or a subset of existing entities in the system. In fact, context type represents a contextual attribute of the system; e.g. time or location of entities. Formally, a context type $ct \in ContextSet$ is defined as a 5-tuple:

$$ct = \langle ValueSet_{ct}, OperatorDefinerSet_{ct}, RelatorSet_{ct}, EntityTypeSet_{ct}, LURSet_{ct}\rangle$$

More detail on each component of the context type *ct* is given below.

3.3.1 Set of Admissible Values: ValueSet$_{ct}$

$ValueSet_{ct}$ denotes the set of values that can be assigned to variables of context type *ct*. Set representation can be used to determine members of $ValueSet_{ct}$. For instance, the value set of context type *time* can be defined in the following way using *set comprehension:* $ValueSet_{time} = \{n : \quad | \ 0 \le n \le 24\}$.

3.3.2 Operator Definer Set: OperatorDefinerSet$_{ct}$

$OperatorDefinerSet_{ct}$ is comprised of a finite number of functions each of which defines logical, set and other user-defined operators on the value set of context type *ct*. Each of these functions requires three arguments, but the types of these arguments are different among the functions. Generally speaking, each *Operator-Definer*$_{ct}$ determines that for two arbitrary values *A* and *B* related to $ValueSet_{ct}$ and $op \in$ *a subset of OperatorSet* whether *(A op B)* is true or not. Since the signature of each *Operator-Definer* function is unique, the signature must be included along the definition. The informal signature of *Operator-Definer* function is as follows:

Operator-Definer$_{ct}$*: A set of values related to* $ValueSet_{ct}$ $\times$ *a set of operators* $\times$ *A set of values related to* $ValueSet_{ct}$ $\rightarrow$ *{true, false}*

For some context types, the specification of an *Operator-Definer* function might be complex. There exist two alternatives for definition of *Operator-Definer* function. First, it can be specified using *propositional logic* and second, it can be incorporated into model using an external module. The detail is omitted due to lack of space.

3.3.3 Set of Admissible Relators: RelatorSet$_{ct}$

$RelatorSet_{CT}$ represents the set of admissible relators for context type *CT*. For instance, for context type *location*, $RelatorSet_{location}$ can be defined as follows:

$$RelatorSet_{location} = \{Is, Entering, Leaving\}$$

3.3.4 Set of Admissible Entity Types: EntityTypeSet$_{ct}$

$EntityTypeSet_{ct}$ denotes the set of entity types related to context type *ct*. In addition, the value set of other context types can be included in $EntityTypeSet_{ct}$ and it simply means that a context type might express a property about a value of another context type. In fact, $EntityTypeSet_{ct}$ is a subset of the set $\{Subject, Object, Environment, ValueSet_{ct_1}, \ldots , ValueSet_{ct_n}\}$. As an example, context type location represents a property which is only related to users, subjects and objects and therefore:

$$EntityTypeSet_{Location} = \{User, Subject, Object\}$$

As another example, consider a context type *locationlvl* which associates a confidentiality level with each value of context type *location*. Then:

$$EntityTypeSet_{locationlvl} = \{ValueSet_{location}\}$$

3.3.5 Level Update Rules: LURSet$_{ct}$

Each *level update rule* (LUR) describes how confidentiality or integrity levels of users, subjects and objects are updated based on their contextual values for context type *ct*. Informally, a $LUR \in LURSet_{ct}$ is a state machine in which confidentiality or integrity levels represent states and 'conditions on contextual values' corresponds to transitions. When a contextual value of context type *ct* related to an entity changes, the conditions are evaluated and entity's (confidentiality or integrity) level is updated based on the result of evaluation.

$LURSet_{ct}$ denotes a set which itself is comprised of two sets of LURs: confidential level update rule set or $C\text{-}LURSet_{ct}$ and integral level update rule set or $I\text{-}LURSet_{ct}$.

$C\text{-}LURSet_{ct}$ includes confidential level update rules of type ct ($C\text{-}LUR_{ct}$). A $C\text{-}LUR_{ct}$ specifies how confidentiality level of entities is updated based on changes in context predicates of type ct.

The confidential level update rules of $C\text{-}LURSet_{ct}$ are generally divided into four categories. The first, second and third categories includes $C\text{-}LUR_{ct,USR}$, $C\text{-}LUR_{ct,SBJ}$, and $C\text{-}LUR_{ct,OBJ}$ respectively. Each of these rules defines a level update rule for confidentiality level of users/subjects/objects based on changes in their contextual value for context type *ct*. The fourth category includes a group of *C-LUR*s in the form of $C\text{-}LUR_{ct,en}$. Each of these *LUR*s defines a level update rule for confidentiality level of a special entity. For instance, $C\text{-}LUR_{ct,en}$ defines how confidentiality level of an entity *en* changes based on its contextual value for context type *ct*. It is evident that if $C\text{-}LURSet_{ct}$ contains a specialized *C-LUR* for an entity, it overrides the general *C-LUR*s defined in other categories. Notice that inclusion of these categories in $C\text{-}LURSet_{ct}$ is optional and $C\text{-}LURSet_{ct}$ might be even empty.

I-LURSet$_{ct}$ includes integral level update rules of context type *ct* (*I-LUR$_{ct}$*). An *I-LUR$_{ct}$* specifies how integrity level of entities is updated based on changes in context predicates of type *ct.* The integral level update rules of *I-LURSet$_{ct}$* are generally divided into four categories as defined for *C-LURSet$_{ct}$*. As above, inclusion of these categories in *I-LURSet$_{ct}$* is optional and *I-LURSet$_{ct}$* might be even empty.

Confidential/Integral Level Update Rule: C-LUR$_{ct}$, I-LUR$_{ct}$. As mentioned earlier, each LUR is simply a state machine. Also, LURs are divided into two categories: C-LURs and I-LURs. For a C-LUR, ConfLvl denotes the set of states and for an I-LUR, IntegLvl constitutes this set. The transitions, on the other hand, are simply some conditions on contextual values of entities for context type ct.

For an *LUR* to act in a correct way, we need to store the previous confidentiality/integrity levels of an entity, before applying that *LUR* to it. The reason for such need will be explained later. Specifically, we need two extra variables for every pair of (entity, context type). For a pair like (*en*, *ct*), these variables are represented by $\lambda_{ct}(en)$ and $\omega_{ct}(en)$ and are initialized in the following way:

$$\forall ct \in ContextSet\ \forall en \in (Subject \cup Object \cup User).\lambda_{ct}(en) = \lambda(en) \wedge \omega_{ct}(en) = \omega(en)$$

Each transition is composed of a set of statements each of which is a conjunction of two conditions: one on contextual value and one on previous confidentiality/integrity levels. The transition takes place if all conditions of all statements are evaluated to true. For instance, suppose in a *C-LUR$_{ct}$* the following transitions is defined:

$$\{(Is, \geq, 10, (=,TS)),(Is,\leq 20,(=,TS))\}$$

This transition takes place if the following statement is evaluated to true:

$$(ct[en][Is] \geq 10 \wedge \lambda_{ct}(en) = TS) \wedge (ct[en][Is] \leq 20 \wedge \lambda_{ct}(en) = TS)$$

Furthermore, the second condition is optional and can be equal to $(\perp, \perp)$; since sometimes there is no restriction on the previous confidentiality/integrity level.

Due to lack of space, the formal definition of a level update rule is omitted here. Instead, an example is used to clarify the concept. Assume $ConfLvl = \langle TS,S,C,U \rangle$. Fig. 1 shows *C-LUR$_{Age,OBJ}$* that describes how objects' confidentiality level is updated based on their Age.

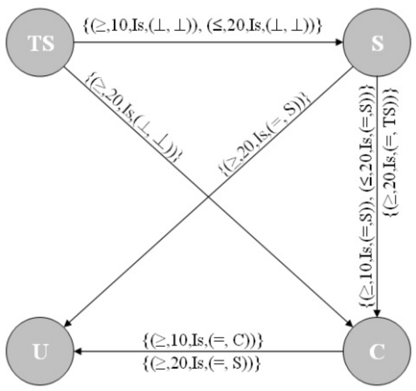

Fig. 1. Confidential level update rules of context type *Age* for objects: *C-LUR$_{Age,OBJ}$*

$C\text{-}LUR_{Age,OBJ}$ simply specifies that the confidentiality level of an object decreases every decade with the restriction that the confidentiality level of an object can never decrease more than two levels. In particular, assume a document named *Doc* is 10 to 20 years old and $\lambda_{Age}(Doc) = \lambda(Doc) = S$. When $C\text{-}LUR_{Age,OBJ}$ is applied to *Doc* for the first time, the transition $(\{(Is,\geq,10,(=,S)), (Is,\leq,20, (=,S))\})$ is evaluated to true and therefore, $\lambda_{Age}(Doc)$ = S, $\lambda(Doc) = C$. In other words, the above transition denotes the following conditional statement:

$$(Age[Doc][Is] \geq 10 \wedge \lambda_{Age}(Doc) = S) \wedge (Age[Doc][Is] \leq 20 \wedge \lambda_{Age}(Doc) = S)$$

As long as the age of *Doc* is between 10 and 20, application of $C\text{-}LUR_{Age,OBJ}$ on *Doc* causes no change in levels, since none of the transitions from state *C* to *U* are evaluated to true. When its age is changed to above 20, the transition $(\{(\geq,20,Is,(=,S))\})$ is evaluated to true and the following assignments takes place:

$$\lambda_{Age}(Doc) = C,\ \lambda(Doc) = U$$

Algorithms for Applying LURs to Entities. In this section, algorithms for applying LURs to entities are presented. To reduce the complexity, we propose two algorithms: one for C-LURs and one for I-LURs.

```
Apply-CLUR ($ct \in ContextSet$, $I \in C\text{-}LURSet_{ct}$, $e \in EntitySet \setminus \{environment\}$){
$\lambda_{ct}(e) = \lambda(e)$
For each state s in ConfLvl
        For each transition from $\lambda(e)$ with label $\{(co_1,v_1,r_1,P_1),\ldots,(co_n,v_n,r_n,P_n)\}$
    to s in I
                if (($P_1 = (\perp,\perp)$ AND Operator-Definer$_{ct}$($ct[e][r_1]$, $co_1$, $v_1$)) OR
                ($P_1 = (do_1,l_1)$ AND Operator-Definer$_{ct}$($ct[e][r_1]$,$co_1$,$v_1$) AND
            $\lambda_{ct}(e)$ $do_1$ $l_1$))
                AND
                ...
                AND
                if (($P_n = (\perp,\perp)$ AND Operator-Definer$_{ct}$($ct[e][r_n]$, $co_n$, $v_n$)) OR
                ($P_n = (do_n,l_n)$ AND Operator-Definer$_{ct}$($ct[e][r_n]$, $co_n$, $v_n$) AND
            $\lambda_{ct}(e)$ $do_n$ $l_n$))
                    $\lambda(e) = s$
}
```

In order to preserve the confidentiality level of entity before being changed, $\lambda(e)$ is assigned to $\lambda_{ct}(e)$. Next, each transition from state $\lambda(e)$ to all other states is evaluated. If the result of evaluation for a transition to a state *s* is true, *s* is assigned to $\lambda(e)$. If none of the transitions is evaluated to true, $\lambda(e)$ is not changed.

Furthermore, for every statement (co_i,v_i,r_i,P_i) of a transition, if $P_i = (\perp,\perp)$, then only the first condition will be evaluated ($Operator\text{-}Definer_{ct}(ct[e][r_i], co_i, v_i)$). But if $P_i \neq (\perp,\perp)$ both conditions will be evaluated:

$$Operator\text{-}Definer_{ct}(ct[e][r_i], co_i, v_i)\ AND\ \lambda_{ct}(e)\ do_i\ l_i$$

To apply $C\text{-}LUR_{ct}$ to an entity *en*, *Apply-CLUR* will be called in the following way:

$$Apply\text{-}CLUR(ct, C\text{-}LUR_{ct}, en)$$

Since the algorithm for applying *I-LUR*s to entities has minor changes compared to *Apply-CLUR* (λ substituted with ω and *ConfLvl* substituted with *IntegLvl*), we omit the details here.

The Reason for Storing Previous Levels of Entities. As mentioned above, we need two extra variables for each pair of (e, ct) where $e \in EntitySet \setminus \{environment\}$ and $ct \in ContextSet$: one for storing previous confidentiality level of the entity before being changed by one of C-LURs of ct and one for storing its previous integrity level before being changed by one of I-LURs of ct. They are represented by $\lambda_{ct}(en)$ and $\omega_{ct}(en)$ respectively.

Since, *LUR*s are applied to entities on special occasions, for a change in context, it is impossible to find out whether an *LUR* has already been applied to an entity or not. In order words, when a change occurs in context, there must be a way to recognize whether this change has already been considered or not. These extra variables are needed to for this matter. Further detail on this issue is omitted due to lack of space.

An Algorithm for Updating Levels of an Entity. UpdateEntityLevels updates the confidentiality and integrity levels of a specific entity (passed to it as an argument) based on the appropriate LURs of all context types in ContextSet.

```
UpdateEntityLevels($e \in EntitySet \setminus \{environment\}$, $ET \in \{USR,SBJ,OBJ\}$){
for each context type $ct \in ContextSet$ in order
        if $C\text{-}LUR_{ct,e} \in C\text{-}LURSet_{ct}$
                Apply-CLUR(ct, $C\text{-}LUR_{ct,e}$, e)
        else if $C\text{-}LUR_{ct,ET} \in C\text{-}LURSet_{ct}$
                Apply-CLUR(ct, $C\text{-}LUR_{ct,ET}$, e)
        if $I\text{-}LUR_{ct,e} \in I\text{-}LURSet_{ct}$
                Apply-ILUR(ct, $I\text{-}LUR_{ct,e}$, e)
        else if $I\text{-}LUR_{ct,ET} \in I\text{-}LURSet_{ct}$
                Apply-ILUR(ct, $I\text{-}LUR_{ct,ET}$, e)
}
```

In this algorithm, *ET* represents the type of Entity. *USR*, *SBJ*, and *OBJ* represent *User*, *Subject* and *Object* sets respectively. The *LUR*s of context types are applied based on the ordering defined by *ContextSet*; i.e. the first element of the ordered set is applied first and so forth. For each context type *ct,* it first checks if there is a specific *C-LUR* defined for entity *e (If* $C\text{-}LUR_{ct,e} \in C\text{-}LURSet_{ct}$) and if so, the *C-LUR* is applied to the entity. Otherwise, it checks if there is a general *C-LUR* based on the type of entity (*else if* $C\text{-}LUR_{ct,ET} \in C\text{-}LURSet_{ct}$) to be applied to it. The same procedure is adopted for *I-LUR*s.

3.4 Operations

3.4.1 AccessRightSet$_{opr}$

The set of access rights in CAMAC model is comprised of *read* and *write.* In CAMAC, every operation, based on what it carries out, includes a subset of these modes; e.g. if it only does an observation of information and no alteration, it only includes *read* and so on. $AccessRightSet_{opr}$ is a subset of the set {*read, write*} which denotes access right set of the operation.

3.4.2 Constraint$_{opr}$

Each operation includes a *constraint* which denotes the prerequisite conditions that must be satisfied before the operation is executed. For *opr* ∈ *OperationSet*, this constraint is represented by *Constraint$_{opr}$* and is mainly composed of *condition blocks.* There exist three types of condition blocks: Confidential condition blocks (*C-CB*), Integral condition blocks (*I-CB*) and Contextual condition blocks (*Cxt-CB*). In defining each condition block, we make use of variable *USR, SBJ* and *OBJ* to represent user, subject and object respectively. Use of these variables allows us to define generic constraints. Next, we define different types of condition blocks and later a grammar for derivation of constraints is presented.

Confidential Condition Block (C-CB). A confidential condition block is defined as a triple $\langle \lambda_1, op, \lambda_2 \rangle$ *in which* $\lambda_1, \lambda_2 \in$ *ConfLvl and op* ∈ *DomOperatorSet. For instance* $\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle$ *is a C-CB denoting the simple security property of Bell-LaPadula.*

Integral Condition Block (I-CB). An integral condition block is defined as a triple $\langle \omega_1, op, \omega_2 \rangle$ *in which* $\omega_1, \omega_2 \in$ *IntegLvl and op* ∈ *DomOperatorSet. For instance* $\langle \omega(SBJ), \geq, \omega(OBJ) \rangle$ *is an I-CB denoting the integrity *-property of Biba.*

Contextual Condition Block (Cxt-CB). A contextual condition block is defined as a triple $\langle Value_1, op, Value_2 \rangle_{ct}$ *in which* $Value_1, Value_2 \in ValueSet_{ct}\{.element\}$, *ct* ∈ *ContextSet and op* ∈ *OperatorSet. The subscript ct determines that operator definer functions of context type ct must be used to evaluate this Cxt-CB. Instances of Cxt-CB are* $\langle Time[environment][Is], <, 9 \rangle_{Time}$ *and* $\langle Age[SBJ][Is], >, Age[OBJ][Is] \rangle_{Age}$.

A Grammar for Derivation of Constraints. Constraints are built using the following unambiguous grammar:

$$
\begin{array}{l}
Constraint \rightarrow Constraint \vee C1 \\
Constraint \rightarrow C1 \\
C1 \rightarrow C1 \wedge C2 \\
C1 \rightarrow C2 \\
C2 \rightarrow (Constraint) \\
C2 \rightarrow Cxt\text{-}CB \mid C\text{-}CB \mid I\text{-}CB
\end{array}
$$

For example, for an operation named *GenerateReport* the following constraint may be defined using the above grammar:

$$Constraint_{GenerateReport} = (\langle \lambda(SBJ), \geq, S \rangle) \vee (\langle \lambda(SBJ), =, C \rangle \wedge \langle Time[environment][Is], \geq, 6 \rangle_{Time} \wedge \langle Time[environment][Is], \leq, 12 \rangle_{Time})$$

Definition of operations finalizes specification of elements of CAMAC model. Next we consider how requests are authorized in CAMAC.

3.5 Authorization of Action

A subject's request to access an object is represented by an *action.* Formally, an action *A* is a triple $\langle s, o, opr \rangle$ in which s ∈ *Subject*, o ∈ *Object* and *opr* ∈ *Operation*. Furthermore the user of an action is the user on behalf of whom the subject is acting; i.e. *u = RepOf(s).* The algorithm *AuthorizeAction* handles authorization of actions.

```
$AuthorizeAction(A = \langle s, o, opr \rangle)$
{
        $u = RepOf(s)$
        $Constraint_A = Constraint_{opr}$
        $UpdateEntityLevels(u, USR)$
        $UpdateEntityLevels(s, SBJ)$
        $UpdateEntityLevels(o, OBJ)$
        $\lambda(s) = GLB(\lambda(s), \lambda(u))$
                $\omega(s) = GLB(\omega(s), \omega(u))$
        $if\ Read \in AccessRightSet_{opr}$
                $Constraint_A = Constraint_A \wedge (\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle \wedge \langle \omega(OBJ), \geq, \omega(SBJ) \rangle)$
        $if\ Write \in AccessRightSet_{opr}$
                $Constraint_A = Constraint_A \wedge (\langle \lambda(SBJ), \leq, \lambda(OBJ) \rangle \wedge \langle \omega(OBJ), \leq, \omega(SBJ) \rangle)$

        Assign u, s, o to USR, SBJ, OBJ in $Constraint_A$ respectively
        return $Evaluate(Constraint_{opr})$
}
```

Upon occurrence of an action, initially the confidentiality and integrity levels of user, subject and object of an action must be updated. As mentioned in section 3.3, *UpdateEntityLevels* algorithm updates the levels of an entity using all the applicable *LUR*s of all context types. Calling the algorithm for user, subject and object takes care of these updates. Since the confidentiality and integrity levels of a subject must be dominated by the corresponding levels of its user, after updating levels of user and subject, the following assignments seems indispensable:

$$\lambda(s) = min(\lambda(s), \lambda(u)),\ \omega(s) = min(\omega(s), \omega(u))$$

After level updates are done, the constraint of the action must be evaluated. Constraint of an action *A* is represented by $Constraint_A$ and is initially equal to operation constraint. Before evaluation takes place, the corresponding confidentiality and integrity constraints must be added to the constraint of action based on access right set of the operation. In other words, if $read \in AccessRightSet_{opr}$, simple security property of Bell-LaPadula and simple integrity property of Biba must be added to $Constraint_A$

$$read \in AccessRightSet_{opr}. Constraint_A = Constraint_A \wedge (\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle \wedge \langle \omega(OBJ), \geq, \omega(SBJ) \rangle)$$

Also, if $write \in AccessRightSet_{opr}$, *-property of Bell-LaPadula and integrity *-property of Biba must be added to $Constraint_A$.

$$write \in AccessRightSet_{opr}. Constraint_A = Constraint_A \wedge (\langle \lambda(OBJ), \geq, \lambda(SBJ) \rangle \wedge \langle \omega(SBJ), \geq, \omega(OBJ) \rangle)$$

At last, *u*, *s* and *o* are assigned to *USR*, *SBJ* and *OBJ* respectively and the constraint is evaluated using a parser, operator definer functions of context types, and dominance relationship. If the result of evaluation is true, the action is granted and otherwise denied.

4 CAMAC Expressiveness

Various mandatory concepts can be expressed using CAMAC. In this paper, due to lack of space, we only express set of categories with it, while some famous models and policies such as Dion and Chinese Wall can conveniently be expressed by the model.

The confidentiality levels in the original Bell-LaPadula model are defined by two components: a classification and a set of categories. On the other hand, as defined in section 3.1 the confidentiality levels of CAMAC model consists of the first component and the set of categories is simply ignored. The same statement holds for integrity levels of Biba. We intend to show that the set of categories is inherently a contextual concept and can be simply modeled as a context type. Here, we take confidentiality levels into consideration. The set of categories for integrity levels can be modeled in a similar way.

The set of categories is a subset of a non-hierarchical set of elements and the elements of this set depend on considered environment and refer to the application area to which information pertains or where data is to be used. A classic example of this set is {*Nato, Nuclear, Crypto*} which denotes the categories in which the classification of the confidentiality level is defined. We define a context type *C-Category* as follows:

$C\text{-}Category = \langle ValueSet_{C\text{-}Category}, OperatorDefinerSet_{C\text{-}Category}, RelatorSet_{C\text{-}Category}, EntityTypeSet_{C\text{-}Category}, LURSet_{C\text{-}Category}\rangle$

- $ValueSet_{C\text{-}Category} = \{P(\{Nato, Nuclear, Crypto\})\}$
- $OperatorDefinerSet_{C\text{-}Category}$

```
    {
        $Operator\text{-}Definer_{C\text{-}Category}(A \in ValueSet_{C\text{-}Category}, o \in OperatorSet, B \in ValueSet_{C\text{-}Category})\{$
            $(A = \{Nato\} \wedge B = \{Nato, NuClear\} \wedge o = '\subset') \vee \ldots$
        }
    }
```

- $RelatorSet_{C\text{-}Category} = \{Is\}$
- $EntityTypeSet_{C\text{-}Category} = \{User, Subject, Object\}$
- $LURSet_{C\text{-}Category} = \{C\text{-}LURSet_{C\text{-}Category}, I\text{-}LURSet_{C\text{-}Category}\}$
  - $C\text{-}LURSet_{C\text{-}Category} = \phi$, $I\text{-}LURSet_{C\text{-}Category} = \phi$

Now the constraints of all operations in *OperationSet* are changed in the following way:

$\forall\ opr \in OperationSet \mid read \in AccessRightSet_{opr}.$
$Constraint_{opr} = (Constraint_{opr}) \wedge (\langle C\text{-}Category[OBJ][Is], \subseteq, C\text{-}Category[SBJ][Is]\rangle_{C\text{-}Category})$

$\forall\ opr \in OperationSet \mid write \in AccessRightSet_{opr}.$
$Constraint_{opr} = (Constraint_{opr}) \wedge (\langle C\text{-}Category[SBJ][Is], \subseteq, C\text{-}Category[OBJ][Is]\rangle_{C\text{-}Category})$

Assume $opr \in OperationSet$ and $read \in AccessRightSet_{opr}$. Based on definition, a confidentiality level $L_1 = (c_1, s_1)$ is higher or equal to (dominates) level $L_2 = (c_2, s_2)$ if and only if the following relationships are valid: $c_1 \geq c_2$, $s_1 \supseteq s_2$

Notice that an action $A = \langle s, o, opr\rangle$ is authorized if the following condition blocks are true: $\langle \lambda(s), \geq, \lambda(o)\rangle$, $\langle C\text{-}Category[s][Is], \supseteq, C\text{-}Category[o][Is]\rangle$

These condition blocks denote aforementioned relationships and since both of them must be satisfied for an action including *opr* to be authorized, it has the same effect as incorporating set of categories in confidentiality levels.

5 Evaluation and Conclusion

CAMAC model could be evaluated and compared with other mandatory models on plenty of basis: authorization time complexity, complexity of policy description, support for context-awareness, expressiveness and security objective. Here we only consider time complexity of authorization due to lack of space.

One important metric would be the computational time needed to authorize an action.

It can be shown that for the computational time to be polynomial, the maximum of time complexities of all *Operator-Definer* functions, must be polynomial. This assumption may not be necessarily true in all cases. Specifically, if the function is added as an external module to the system, there is no guarantee in this regard.

In this paper, we explained the need for a context-aware mandatory access control model and presented CAMAC as a model which satisfies such a need. CAMAC model utilizes context-awareness to provide dynamicity and context-sensitivity of levels to enable specification of sophisticated mandatory policies. In addition, various mandatory controls can be incorporated into the CAMAC model. Bell-LaPadula and Biba strict integrity policy are the inbuilt part of the model and other Biba policies, Chinese Wall policy and Dion can be appended to the model using context types. Also, an amalgamation of mandatory policies can be used simultaneously. For instance, Bell-LaPadula, Biba strict integrity policy, and Chinese Wall Policy can all be deployed at once.

References

1. Bell, D.E., LaPadula, L.J.: Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997 Rev. 1. MITRE Corporation (1976)
2. Bell, D.E., LaPadula, L.J.: Secure Computer Systems: Mathematical Foundations. Technical Report MTR-2547. MITRE Corporation (1976)
3. Biba, K.: Integrity Considerations for Secure Computer Systems. In: Corporation, M. (ed.): Technical Report MTR-3153, Bedford, MA (1977)
4. Dion, L.C.: A Complete Protection Model. In: IEEE Symposium on Security and Privacy, Oakland, CA, pp. 49–55 (1981)
5. Brewer, D.F.C., Nash, M.J.: The Chinese Wall Security Policy. In: IEEE Symposium Research in Security and Privacy, pp. 215–228. IEEE CS Press, Los Alamitos (1989)
6. Sandhu, R.S.: Lattice-Based Access Control Models. IEEE Computer 26(11), 9–19 (1993)
7. Sandhu, R.S., Samarati, P.: Access Controls: Principles and Practice. IEEE Communications 32 (9), 40–48 (1994)
8. Kumar, A., Karnik, N., Chafle, G.: Context Sensitivity in Role Based Access Control. ACM SIGOPS Operating Systems Review, 53–66 (2002)
9. Al-Kahtani, M.A., Sandhu, R.: A Model for Attribute-Based User-Role Assignment. In: 18th Annual Computer Security Applications Conference, pp. 353–364. IEEE Computer Society Press, Las Vegas (2002)

10. Covington, M., Moyer, M., Ahamad, M.: Generalized role-based access control for securing future applications. In: 23rd National Information Systems Security Conference, Baltimore, MD, USA (2000), http://csrc.nist.gov/nissc/2000/proceedings/toc.pdf
11. Zhang, G., Parashar, M.: Context-aware dynamic access control for pervasive applications. In: Communication Networks and Distributed Systems Modeling and Simulation conference, San Diego (2000)
12. Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K.: Flexible Team-based Access Control Using Contexts. In: Sixth ACM Symposium on Access Control Models and Technologies, pp. 21–27. ACM Press, Chantilly (2001)
13. Hu, J., Weaver, A.C.: A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In: First Workshop on Pervasive Privacy Security, Privacy, and Trust, Boston, MA, USA (2004), http://www.pspt.org/techprog.html
14. Ray, I., Kumar, M.: Towards a location-based mandatory access control model. Computers & Security 25, 36–44 (2006)
15. Baldauf, M., Dustdar, S.: A Survey on Context-aware Systems. Technical report TUV-1841-2004-24. Distributed Systems Group, Technical University of Vienna (2004)
16. Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., Malm, E.-J.: Managing Context Information in Mobile Devices. IEEE Pervasive Computing 2 (3), 42–51 (2003)
17. Tao Gu, X.H.W., Pung, H.K., Zhang, D.Q.: A Middleware for Building Context-Aware Mobile Services. In: IEEE Vehicular Technology Conference, Milan, Italy, vol. 5, pp. 2656–2660 (2004)
18. Fahy, P., Clarke, S.: CASS: Middleware for Mobile, Context-Aware Applications. In: Workshop on Context Awareness at MobiSys., Boston, pp. 304–308 (2004)
19. Chen, H., Finn, T., Joshi, A.: Using OWL in a Pervasive Computing Broker. In: Workshop on Ontologies in Open Agent Systems, AAMAS 2003, Melbourne, Australia, pp. 9–16 (2003)
20. Dey, A.K., Salber, D., Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction (HCI) Journal 16(2-4), 97–166 (2001)