The ISC Int'l Journal of Information Security

January 2009, Volume 1, Number 1 (pp. 35–54)

http://www.isecure-journal.org

CAMAC: A Context-Aware Mandatory Access Control Model

Jafar Haadi Jafarian [a,*], Morteza Amini [a]

[a] *Sharif Network Security Center, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

ARTICLE INFO

Article history:
Received: 26 April 2008
Revised: 4 August 2008
Accepted: 18 August 2008
Published Online: 28 January 2009

Keywords:
Mandatory Access Control,
Multilevel Security, Authorization,
Context Awareness,
Information Flow Control

ABSTRACT

Mandatory access control models have traditionally been employed as a robust security mechanism in multilevel security environments such as military domains. In traditional mandatory models, the security classes associated with entities are context-insensitive. However, context-sensitivity of security classes and flexibility of access control mechanisms may be required especially in pervasive computing environments. To this aim, we propose a context-aware mandatory access control model (CAMAC) capable of dynamic adaptation of access control policies to context, and of handling context-sensitive class association, in addition to preservation of confidentiality and integrity as specified in traditional mandatory access control models. In order to prevent any ambiguity, a formal specification of the model and its elements such as context predicates, context types, level update rules, and operations is required. High expressiveness of the model allows specification of the traditional mandatory access control models such as BLP, Biba, Dion, and Chinese Wall. The model can also be considered as an information flow control model with context-sensitive association of security classes.

1 Introduction

Access control models govern the access of users to information based on some specified rules. The purpose of access control is to limit actions or operations that a legitimate user of a computer system can perform [1]. As computing technology becomes more pervasive and mobile services more deployed, applications will need flexible access control mechanisms. Unlike traditional approaches to access control, access decisions for these applications 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. However, they have been deployed in commercial sectors as well [2]. Mandatory access control is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity [3]. Specifically, security classes are associated with every subject and object in the system; and, access of a subject to an object is granted if some axioms specified over the security classes of subjects and objects are satisfied [4].

Numerous context-aware access control models have been proposed in the literature. However, none of the models directly target new security require-

* Corresponding author.
Email addresses: jafarian@ce.sharif.edu (J.H. Jafarian), m_amini@ce.sharif.edu (M. Amini).

ISSN: 2008-2045

ments of multilevel environments. 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, security classes associated with entities are usually permanent and insensitive to context. However, in some systems, we may need dynamic and context-sensitive association of security classes. As an example, in most intelligence agencies, the security level of documents decreases after a certain time. Moreover, in the forthcoming pervasive computing environment, 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 propose the Context-Aware Mandatory Access Control (CAMAC) model capable of dynamic adaptation of policies with context and handling context-sensitive class association while preserving confidentiality and integrity. In fact, CAMAC uses Bell-LaPadula and Biba properties to preserve confidentiality and integrity of information.

Mandatory access control models have traditionally been used in information flow control; and, due to the context-sensitive class association in CAMAC, the model can conveniently be considered as an information flow control model with context-sensitive security levels.

The rest of the paper is organized as follows. In the next section, rudimentary concepts of mandatory access control and context-awareness are reviewed. In section 3, related work is surveyed. Section 4 formally introduces the CAMAC model. Section 5 provides a simple system based on CAMAC and explains its function in an access scenario. In section 6, the expressiveness of CAMAC is scrutinized. In section 7, we show that the CAMAC model is a lattice-based model; and, due to its context-sensitive security classes, it can be considered as an information flow model in which security classes associated with entities are context-sensitive. Finally, CAMAC is evaluated and compared to other mandatory models in various aspects, followed by some conclusions.

2 Preliminaries

2.1 Mandatory Access Control

Numerous mandatory access control models and policies have been introduced in literature, most important of which are Bell-LaPadula [5,6], Biba [7], Dion [8], and Chinese Wall [2]. Since the Bell-LaPadula and Biba models are explicitly used in CAMAC, a brief overview of them is presented here. The Chinese Wall policy will be further explored in section 6.2.

The Bell-LaPadula (BLP) model aims at preserving confidentiality and preventing unauthorized disclosure of information. Each entity (subject or object) has a confidentiality [1] level defined by two components: a classification and a set of categories. The confidentiality levels with dominance ($\geq$) relationship form a lattice ($c_1 \geq c_2$, $s_1 \supseteq s_2$). Let function λ maps an entity (subject or object) into its confidentiality level. The principles for reading and writing objects are given by the *Simple Security Property* and the **-Property* as defined below.

Simple Security Property (no read-up): A subject s is allowed to read the object o only if the confidentiality level of the subject, $\lambda(s)$, dominates the confidentiality level of the object, $\lambda(o)$.

***-Property (no write-down)**: A subject s is allowed to write into an object o if the confidentiality level of the object, $\lambda(o)$, dominates the confidentiality level of the subject, $\lambda(s)$ [2].

The Biba model [7] aims at achieving integrity by preventing unauthorized modification of information [3]. Biba proposed several integrity policies where the best known is *strict integrity policy*, which we refer to as the Biba model; and, is specified in terms of subjects, objects, and their integrity levels. Assuming function ω maps an entity into its integrity level, the principles for reading and writing objects are given by the *Simple Integrity Property* and the *Integrity *-Property*, as defined below. The principles are the duals of principles *no read-up* and *no write-down* in the BLP model.

[1] In the original paper, the term *security level* is used. Here, it is substituted with *confidentiality level* to prevent semantic overlap with *integrity level* in the Biba model.

[2] Many practical implementations do not allow write-up (known as a blind write) and use the restricted *-property based on which a subject s is allowed to write into an object o only if $\lambda(s) = \lambda(o)$. Since Biba properties are included in CAMAC, integrity violation in the *-property is of no concern here; and therefore, the original *-property is used.

[3] Bell and LaPadula [5,6] introduced the concept of integrity; but, their approach has serious flaws.

Simple Integrity Property (no read-down integrity): A Subject s is allowed to read object o only if the integrity level of the object, $\omega(o)$, dominates the integrity level of the subject, $\omega(s)$.

Integrity *-Property (no write-up integrity): A subject s is allowed to write into an object o if the integrity level of the subject, $\omega(s)$, dominates the integrity level of the object, $\omega(o)$.

2.2 Context and Context-Awareness

The term *context-aware* first appeared in [9] where context is defined as location, identities of nearby people and objects, and changes to those objects. Such enumerations of context examples were often used in the beginning of context-aware history systems [10].

Pascoe [11] defines context to be the subset of physical and conceptual states of interest to a nearby entity. Dey [12] enumerates context as the user's emotional state, focus of attention, location and orientation, date and time, objects and people in the user's environment.

A well-known definition of context can be found in Dey *et al.* [13] as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object considered relevant to the interaction between a user and an application, including the user and applications themselves".

A context-aware system is a system that adapts according to its location of use, the collection of nearby people and object, as well as changes to those objects over time [9]. Such systems are capable of using context in providing relevant information and/or services to user. In other words, context-aware systems can extract, interpret, and use context information and adapt their functionality to the current context of use [14]. The adaptation process is accomplished without explicit user interference and aims at enhancing functionalities and capabilities of the system.

3 Related Work

Many researches aim at applying context-awareness to the RBAC model. Kumar *et al.* [15] proposed a context-sensitive RBAC model that enabled traditional RBAC to enforce more sophisticated security policies dependent on the context of an attempted operation. Al-kahtani *et al.* [16] proposed the RB-RBAC model, performing role assignment dynamically based on users' attributes or other constraints on roles. GR-BAC [17] incorporates three types of roles; subject roles which correspond to the traditional RBAC roles, object roles to categorize objects, and environment roles to capture environmental or contextual information. Employment of such role types in specification of access control policies incorporates context-awareness into the model. Zhang *et al.* [18] 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 machine. Therefore, a user's role and a role's permission are determined according to the context. Georgiadis *et al.* [19] presented a team-based access control model that is aware of contextual information associated with activities in applications.

Hu *et al.* [20] developed a context-aware access control model for distributed health-care applications. The model defines context type as a property related to every participant in an application when it is running. In simple cases, context type may be a concrete property familiar in everyday life, such as time or location. Context set is defined as a set of context types. By analyzing the system security requirements, application designers determine which context types are to be used to specify access. Context constraint is a regular expression describing contextual requirements. It consists of triples $(ct, op, value)$ in which $ct \in ContextSet$, op is a logical operator, and $value$ is a specific value of ct. Authorization policy specifies which context constraint should be satisfied for a subject to achieve a specific access to an object. When a user wishes to acquire a specific permission on an object, the system evaluates the corresponding authorization policy and grants the access if the evaluation is successful.

Ray *et al.* [21] proposed a location-based mandatory access control model by extending the BLP model with the notion of location. In particular, every location is associated with a security level and the BLP no read-up and no write-down properties are extended by taking security levels of locations into consideration. Based on Baldauf *et al.*'s classification of context-aware systems [10], the location-based mandatory access control model can be categorized as a location-aware system.

However, no general context-aware mandatory access control model has been proposed so far. Since mandatory access control models directly target multilevel security environments in which entities are classified based on their trust and sensitivity, a context-aware mandatory access control model would be the most appropriate choice in addressing new security requirements of such environments.

4 CAMAC: A Context-Aware Mandatory Access Control Model

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. In particular, a subject's access to an object is contingent on contextual constraints as well as confidentiality and integrity properties. As an example, an access control policy might require that some timing restrictions be satisfied for a subject to acquire a read access to an object, e.g. time must be between $8am$ and $1pm$. CAMAC 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 [4]. For instance, confidentiality level of objects might decrease as their lifetime increases (and becomes accessible to less trustworthy subjects). CAMAC also allows such dynamic level associations based on contextual information.

In addition, like most mandatory models, CAMAC includes data structures for specification of discretionary policies. Like BLP, CAMAC uses an access control matrix denoted with D, in which for each subject s and object o, $D[s, o]$ denotes the access rights that s can have on o. For the sake of simplicity, details related to the specification of discretionary policies are omitted.

Before presenting a formal definition of the model, we illustrate the motivation for access control models such as ours through an example application enabled by a pervasive computing infrastructure in a smart building of a military environment. The building has many rooms, including administration offices, conference rooms, training rooms, etc. Sensors in the building can capture, process, and store a variety of information all around the building, the users, and their activities. Pervasive applications in such an environment allow military forces to access resources/information from any location at anytime using mobile devices (PDAs) and wireless networks. While classification is still the basis for all the access control decisions, the user's 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 the time by the manager in charge. Similarly, the server should not allow access if its load is above 80% or if the access is over an insecure link. In such applications, privileges assigned to the user will change as context changes. The above examples illustrate 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 includes environment of the user such as location and time, where the user accesses 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 the 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.

4.1 Formal Definition of CAMAC

CAMAC can be formally defined as 9-tuple $\langle EntitySet, RepOf, ConfLvl, IntegLvl, \lambda, \omega, ContextPredicateSet, ContextSet, OperationSet\rangle$ where

- $EntitySet$ is the set of all entities in the system and is composed of four disjoint sets: $User$, $Subject$, $Object$, and $Environment$. $User$, $Subject$, and $Object$ sets include users, subjects, and objects of the system respectively, which are disjointed. If an entity can play the role of both a subject and an object, two data structures are allocated to it; one in the $Object$ set and the other in the $Subject$ set. The $Environment$ set has only one member called $environment$.
  $EntitySet = User \cup Subject \cup Object \cup Environment$
  $User \cap Subject \cap Object \cap Environment = \emptyset$
- $RepOf$: $Subject \rightarrow User$ assigns to each subject the user who has 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 such as $\langle c_n, c_{n-1}, ..., c_1\rangle$ in which c_n and c_1 are the highest and lowest levels respectively. As in BLP, each user, subject, and object is associ-

[4] In all mandatory access control models, system administrator is allowed to change security classes associated statically. In BLP, a subject can change the confidentiality level of an object under certain conditions. Also, the Lower-water-mark policy in Biba allows the subject integrity level to decrease in some circumstances.

ated with a confidentiality level. It must be noted that there is a difference between BLP and CAMAC in terms of confidentiality level. While BLP 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 6.1, we show that the second component, set of categories, is a contextual information and can be easily incorporated into the model as a context type.

- $IntegLvl$ is a finite ordered set of integrity levels such as $\langle i_n, i_{n-1}, ..., i_1\rangle$ 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 just the classification component.
- λ is a mapping function which associates each user, subject, and object with a confidentiality level; i.e., $\lambda : User \cup Subject \cup Object \rightarrow ConfLvl$
- ω is a mapping function which associates each user, subject, and object with an integrity level; i.e.,
  $\omega : User \cup Subject \cup Object \rightarrow IntegLvl$
- $ContextPredicateSet$ is the set of current context predicates in the system. Each context predicate is a statement about the value of a contextual attribute. More details on context predicates are presented in section 4.2.
- $ContextSet$ 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 as the 5-tuple $ct = \langle ValueSet_{ct}, OperatorDefinerSet_{ct}, RelatorSet_{ct}, EntityTypeSet_{ct}, LURSet_{ct}\rangle$. Context types are explored in section 4.3.
- $OperationSet$ is the set of all operations in the system. An operation can be executed by a subject on an object under specific circumstances; and, may change the state of subject and object. An operation $opr \in OperationSet$ can be formally defined as the pair $opr = \langle AccessRightSet_{opr}, Constraint_{opr}\rangle$. Details of operations are discussed in section 4.4.

Figure 1 represents a schematic representation of the CAMAC model.

4.2 Context Predicate

Each context predicate is a predicate representing a value for a contextual attribute. As in [22] and [23], we define a context predicate $cp \in ContextPredicateSet$ as the 4-tuple $cp = \langle en, ct, r, v\rangle$ where $en \in \{User, Subject, Object, Environment, ValueSet_{ct_1}, ..., ValueSet_{ct_n}\}$, $ct \in ContextSet$, $r \in RelatorSet_{ct}$, $v \in ValueSet_{ct}$, and $ct_1, ..., ct_n \in ContextSet$.

For example, $\langle John, Location, Is, Classroom\rangle$ is a context predicate, which indicates the current location of subject $John$.

Managing and updating context predicates is the responsibility of Context Management Unit (CMU). The implementation details of CMU are beyond the scope of this paper. Context Managing Framework [24], the SOCAM project [25], CASS project [26], CoBrA architecture [27], the Context Toolkit [28], and frameworks presented in projects like Hydrogen [29] and Gaia [22] can be used as an infrastructure in the implementation of CMU.

The value set of a context type can be a function of other primary context types, e.g., a context type $Load$ would be a function of two primary context types: CPU Usage and Bandwidth Load. It is the responsibility of CMU to handle such correlations among context types, and as such not included in the model. In general, we assume that CMU updates $ContextPredicateSet$ according to environment, user, and system changes; 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 the entity E for the context type X and the relator R; i.e., $X[E][R] = V$. If such a context predicate does not exist in $ContextPredicateSet$, we assume that $X[E][R] = \perp$ (read as null).

$$X[E][R] = V, \langle E, X, R, V\rangle \in ContextPredicateSet$$
$$X[E][R] = \perp, \langle E, X, R, V\rangle \notin ContextPredicateSet$$

Furthermore, for $ContextPredicateSet$ to be consistent, we assume $X[E][R]$ is associated with a unique value. That is, $(\langle E, X, R, a\rangle \in ContextPredicateSet \wedge \langle E, X, R, b\rangle \in ContextPredicateSet) \rightarrow a = b$

4.3 Context Type

Informally, a *context type* 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., the time or location of entities. Formally, a context type $ct \in ContextSet$ is defined as 5-tuple $ct = \langle ValueSet_{ct}, OperatorDefinerSet_{ct}, RelatorSet_{ct}, EntityTypeSet_{ct}, LURSet_{ct}\rangle$.

Figure 2 shows a schematic representation of the

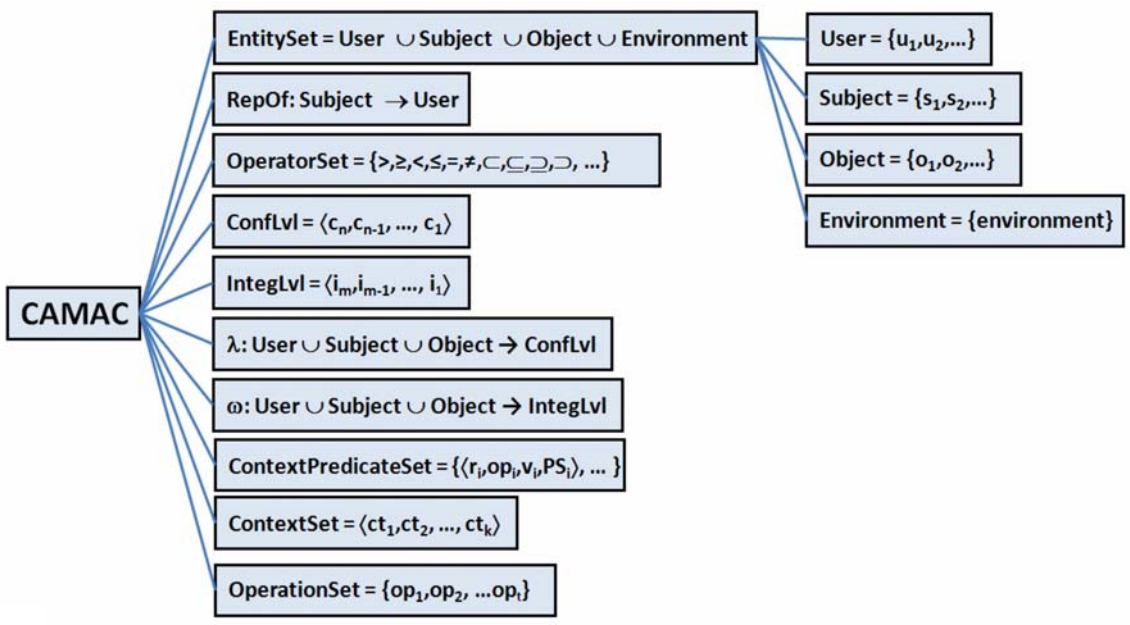

Figure 1. A schematic representation of CAMAC

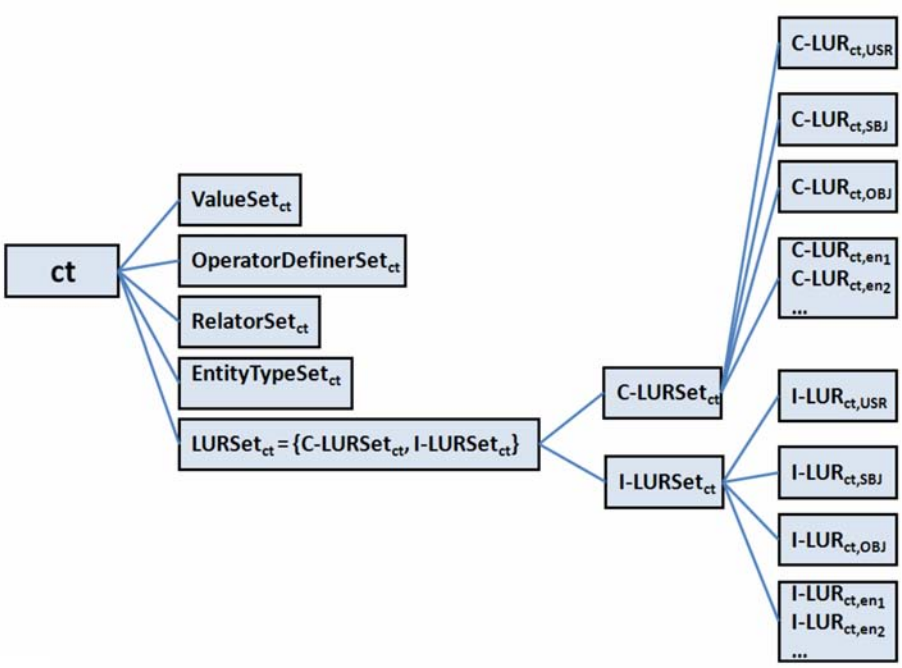

Figure 2. A schematic representation of a context type ct

context type ct. More details on each component of the context type ct are given in the following subsections.

4.3.1 Set of Admissible Values

$ValueSet_{ct}$ denotes the set of values that can be assigned to variables of context type ct. For example, the value set of context type $Time$ can be defined as $ValueSet_{Time} = \{n : \mathbb{N} | 0 \leq n \leq 24\}$.

The value set could be either finite or infinite and either countable or uncountable. Furthermore, we define $\perp$ as undefined value and assume that it belongs to all value sets by default.

4.3.2 Operator Definer Set

$OperatorDefinerSet_{ct}$ is comprised of a finite number of functions, each of which defines user-defined operators on the value set of context

type ct. Each of the functions requires three arguments; but, the types of the arguments are different in different functions. Generally speaking, for two arbitrary values A and B related to $ValueSet_{ct}$ and $op \in a\ subset\ of\ OperatorSet$, each $Operator\text{-}Definer_{ct}$ determines whether $A\ op\ B$ is evaluated as true or not. Since the signature of each $Operator\text{-}Definer$ function is unique, the signature must be included along with the definition. The informal signature of an $Operator\text{-}Definer$ function is as follows:

$Operator\text{-}Definer_{ct} : A\ set\ of\ values\ related\ to\ ValueSet_{ct} \times A\ set\ of\ operators \times A\ set\ of\ values\ related\ to\ ValueSet_{ct} \rightarrow \{true, false\}$

4.3.3 Set of Admissible Relators

$RelatorSet_{ct}$ represents the set of admissible relators for context type ct. For instance, for the context type $location$, $RelatorSet_{Location}$ can be defined as follows: $RelatorSet_{Location} = \{Is, Entering, Leaving\}$

4.3.4 Set of Admissible Entity Types

$EntityTypeSet_{ct}$ denotes the set of entity types related to context type ct. In fact, $EntityTypeSet_{ct}$ is a subset of the set $\{Subject, Object, Environment, ValueSet_{ct_1}, ..., ValueSet_{ct_n}\}$

As an example, the context type $location$ represents a property which is only related to users, subjects, or objects; and therefore: $EntityTypeSet_{Location} = \{User, Subject, Object\}$

On the other hand, the context type $Time$ represents a property which does not concern users, subjects, and objects; and, it is only related to the environment. Therefore: $EntityTypeSet_{Time} = \{Environment\}$

As another example, consider a context type $LocationLvl$, which assigns a confidentiality level to each value of context type Location. Then, we have: $EntityTypeSet_{LocationLvl} = \{ValueSet_{Location}\}$

4.3.5 Level Update Rules

Each level update rule (LUR) describes how confidentiality or integrity levels of users, subjects, and objects are updated based on their contextual values for a typical 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 a context type (related to an entity) changes, the conditions are evaluated and the entity's (confidentiality or integrity) level is updated based on the result of the 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 $I\text{-}LURSet_{ct}$.

Confidential Level Update Rule Set ($C\text{-}LURSet_{ct}$): $C\text{-}LURSet_{ct}$ includes confidential level update rules of type ct ($C\text{-}LUR_{ct}$). A $C\text{-}LUR_{ct}$ specifies how the 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:

(1) The category which includes $C\text{-}LUR_{ct,USR}$ and defines a level update rule for confidentiality level of users based on changes in their contextual value for context type ct.
(2) The category which includes $C\text{-}LUR_{ct,SBJ}$ and defines a level update rule for confidentiality level of subjects based on changes in their contextual value for context type ct.
(3) The category which includes $C\text{-}LUR_{ct,OBJ}$ and defines a level update rule for confidentiality level of objects based on changes in their contextual value for context type ct.
(4) The category which includes a group of $C\text{-}LURs$ in the form of $C\text{-}LUR_{ct,en}$. Each of these $C\text{-}LURs$ defines a level update rule for confidentiality level of a special entity based on changes in its contextual value for context type ct. 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 $C\text{-}LUR$ for an entity, it overrides the general $C\text{-}LURs$ defined in other categories.

Inclusion of the above categories in $C\text{-}LURSet_{ct}$ is optional; and, $C\text{-}LURSet_{ct}$ might be even empty.

Integral Level Update Rule Set ($I\text{-}LURSet_{ct}$): $I\text{-}LURSet_{ct}$ includes integral level update rules of context type ct ($I\text{-}LUR_{ct}$). An $I\text{-}LUR_{ct}$ specifies how integrity level of entities is updated based on changes in the context predicates of type ct. The integral level update rules of $I\text{-}LURSet_{ct}$ are generally divided into four categories which are defined analogous to the categories defined for confidential level update rules. Notice that inclusion of these categories in $I\text{-}LURSet_{ct}$ is optional and $I\text{-}LURSet_{ct}$ might be even empty.

Confidential/Integral Level Update Rule (LUR): As mentioned earlier, each LUR is simply a state machine. Also, $LURs$ are divided into two categories: $C\text{-}LURs$ and $I\text{-}LURs$. For a $C\text{-}LUR$, $ConfLvl$ denotes the set of states; and, for an $I\text{-}LUR$, $IntegLvl$ constitutes the set. The transitions, on the other

hand, are simply some conditions on contextual values of entities for a context type. For a LUR to act in a correct way, we need to store the previous confidentiality/integrity level of an entity before applying the LUR. The reason for such need will be explained later. Specifically, we need two extra variables for every pair of $(entity, context\text{-}type)$. 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 the other one on previous confidentiality/integrity levels. The transition takes place if all conditions of all statements are evaluated as true. For instance, suppose in a $C\text{-}LUR_{ct}$ the following transition is defined: $\{(Is, \geq, 10, (=, TS)), (Is, \leq, 20, (=, TS))\}$

This transition takes place if the following statement is evaluated to be 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. Formally, a $LUR_{ct} \in (C\text{-}LURSet_{ct} \cup I\text{-}LURSet_{ct})$ is defined as the triple $LUR_{ct} = \langle S, V, T \rangle$ where:

- S represents the set of states, and based on the type of LUR, it can be equal to either $ConfLvl$ or $IntegLvl$.
- $V : P(RelatorSet_{ct} \times OperatorSet \times ValueSet_{ct} \times PS)$ defines the alphabet, where $PS : (DomainOprSet \cup \perp) \times (S \cup \perp)$ and $DomainOprSet = \{=, \leq, \geq, <, >\}$; and,
- $T : S \times V \rightarrow S$ defines the transition function.

Through an example, we clarify the definition. Assume $ConfLvl = \langle TS, S, C, U \rangle$. Figure 3 depicts $C\text{-}LUR_{Age,OBJ}$ and describes how objects' confidentiality level is updated based on their Age. $C\text{-}LUR_{Age,OBJ}$ 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 following transition is evaluated to be true:

$(\{(Is, \geq, 10, (=, S)), (Is, \leq, 20, (=, S))\})$
Therefore, $\lambda_{Age}(Doc) = S, \lambda(Doc) = C$. In other words, the above transition denotes the following conditional statement:
$(Age[Doc][Is] \geq 10 \wedge \lambda_{Age}(Doc) = S) \wedge (Age[Doc][Is] \leq 20 \wedge \lambda_{Age}(Doc) = S)$

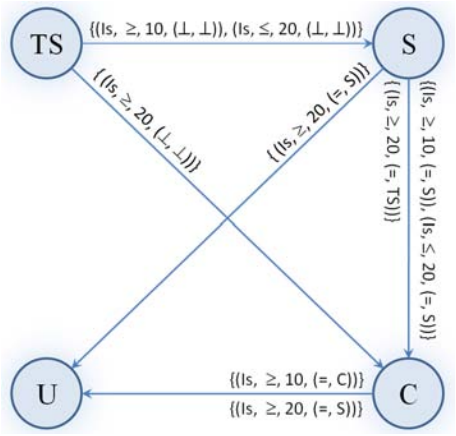

Figure 3. Level update rules of context type Age for objects

As long as the age of Doc is between 10 and 20, the 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 as true. When its age is changed to above 20, the transition $(\{(Is, \geq, 20, (=, S))\})$ is evaluated to true, and thus $\lambda_{Age}(Doc) = C, \lambda(Doc) = U$.

Algorithms for Applying LURs to Entities: We present two algorithms for applying $C\text{-}LURs$ and $I\text{-}LURs$ to entities.

Algorithm 1 is employed to apply a $C\text{-}LUR$ rule to an entity.

Algorithm 1 Apply-CLUR($ct \in ContextSet, I \in C\text{-}LURSet_{ct}, e \in EntitySet \setminus \{environment\}$)

```
1: λ_ct(e) = λ(e)
2: for all state s ∈ ConfLvl do
3:    for all transition from λ(e) with label
      {(r_1, op_1, v_1, PS_1), ..., (r_n, op_n, v_n, PS_n)} to s do
4:       flag = True
5:       for i = 1 to n do
6:          if not {[PS_i = (⊥, ⊥) AND
                    Operator-Definer_ct(ct[e][r_i], op_i, v_i)]
                        OR
                   [PS_i = (do_i, l_i) AND
                    Operator-Definer_ct(ct[e][r_i], op_i, v_i) AND
                    λ_ct(e) do_i l_i]} then
7:             flag = False
8:          end if
9:       end for
10:      if flag = True then
11:         λ(e) = s
12:      end if
13:   end for
14: end for
```

In order to preserve the confidentiality level of an entity before its change, $\lambda(e)$ is assigned to $\lambda_{ct}(e)$. Next, each transition from state $\lambda(e)$ to all other states is evaluated. If the result of the evaluation for

a transition to a state s is true, s is assigned to $\lambda(e)$. If none of the transitions is evaluated to be true, $\lambda(e)$ is not changed.

Furthermore, for every statement (r_i, op_i, v_i, PS_i) of a transition, if $PS_i = (\perp, \perp)$, then only the first condition is evaluated; i.e.,
$(Operator\text{-}Definer_{ct}(ct[e][r_i], op_i, v_i))$.
However, if $PS_i \neq (\perp, \perp)$, then both conditions are evaluated; i.e.,
$Operator\text{-}Definer_{ct}(ct[e][r_i], op_i, v_i)$ AND $ct(e) do_i l_i$.
To apply $C\text{-}LUR_{ct}$ to an entity en, $Apply\text{-}CLUR$ is called as follows: $Apply\text{-}CLUR(ct, C\text{-}LUR_{ct}, en)$.

Analogously, algorithm 2 is employed to apply an $I\text{-}LUR$ rule to an entity.

Algorithm 2 Apply-ILUR($ct \in ContextSet, I \in I\text{-}LURSet_{ct}, e \in EntitySet \setminus \{environment\}$)

```
1: ω_ct(e) = ω(e)
2: for all state s ∈ IntegLvl do
3:    for all transition from ω(e) with label
      {(r_1, op_1, v_1, PS_1), ..., (r_n, op_n, v_n, PS_n)} to s do
4:       flag = True
5:       for i = 1 to n do
6:          if not {[PS_i = (⊥, ⊥) AND
                     Operator-Definer_ct(ct[e][r_i], op_i, v_i)]
                        OR
                    [PS_i = (do_i, l_i) AND
                     Operator-Definer_ct(ct[e][r_i], op_i, v_i) AND
                     ω_ct(e) do_i l_i] then
7:             flag = False
8:          end if
9:       end for
10:      if flag = True then
11:         ω(e) = s
12:      end if
13:   end for
14: end for
```

Since the algorithm has minor changes compared to $Apply\text{-}CLUR$ (λ is substituted with ω and $ConfLvl$ is substituted with $IntegLvl$), we omit the details here.

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

Since, $LURs$ are applied to entities only on special occasions, for a change in context, it is impossible to discover whether a LUR has already been applied to an entity or not. In other words, when a change occurs in context, there must be a way to recognize whether this change has already been considered or not.

Figure 4. $C\text{-}LUR_{ct,USR}$ without considering the second conditions

Figure 5. $C\text{-}LUR_{ct,USR}$ considering the second conditions

To illustrate the point, assume $ConfLvl = \langle s_n, ..., s_1 \rangle$, $ct \in ContextSet$, and $C\text{-}LUR_{ct,USR} \in C\text{-}LURSet_{ct}$. $C\text{-}LUR_{ct,USR}$ stipulates that if a condition like α becomes true, the confidentiality level of the user to which $C\text{-}LUR_{ct,USR}$ is applied must be incremented only once. α denotes a condition on $ct[u][r]$ where $u \in User$ and $r \in RelatorSet_{ct}$.

Assume $C\text{-}LUR_{ct,USR}$ is applied to a user named a for whom $\lambda(a) = s_k$. Without considering the second conditions, assume there is a transition from s_i to s_{i+1} for $i = 1...n-1$ (see Figure 4).

Now if $ct[a][r]$ changes in a way that α becomes true, when $C\text{-}LUR_{ct,USR}$ is applied to a for the first time, $\lambda(a)$ changes to s_{k+1}. Sometimes later, when $C\text{-}LUR_{ct,USR}$ is applied to a for the second time, we have no clue of the fact that $C\text{-}LUR_{ct,USR}$ has already been applied to a and $\lambda(a)$ changes to s_{k+2}.

To solve this problem, the aforementioned variables are used. Assume the condition $\lambda_{ct}(a) = s_i$ is added to every transition from the state s_i to s_{i+1} for $i = 1...n-1$ (see Figure 5).

Now the transition occurs for the first time, since $\lambda_{ct}(a) = s_k$, and $\lambda_{ct}(a)$ and $\lambda(a)$ will become equal to s_{k+1} and s_k respectively. But, the transition does not take place after the second application, since $\lambda_{ct}(a) \neq s_{k+1}$.

The Order of LUR Application: The ordering based on which $LURs$ of different context types are applied to a specific entity influences the final result. To clarify this point, assume $ConfLvl = \langle s_n, ..., s_1 \rangle$, $ContextSet = \langle ct_1, ct_2, ... \rangle$, $C\text{-}LUR_{ct_1,SBJ} \in C\text{-}LURSet_{ct_1}$, and $C\text{-}LUR_{ct_2,SBJ} \in C\text{-}LURSet_{ct_2}$. Also assume that in $C\text{-}LUR_{ct_1,SBJ}$ there is a transition α from s_i to s_j and a transition β from s_m to s_n (see the left side of Figure 6), and in $C\text{-}LUR_{ct_2,SBJ}$, there is a transition β from s_j to s_k and a transition α from s_i to s_m (see the right side of Figure 6). Also $John\text{-}Proc \in Subject$ and $\lambda(John\text{-}Proc) = s_k$.

Suppose at some moment, α and β are true. If $C\text{-}LUR_{ct_1,SBJ}$, is applied to $John\text{-}Proc$ before $C\text{-}LUR_{ct_2,SBJ}$, the final value of $\lambda(John\text{-}Proc)$ is equal to s_k. But, if they are applied in the reverse order, its final value is equal to s_n. As you can see,

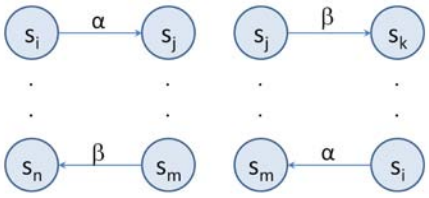

Figure 6. (left) $C\text{-}LUR_{ct_1,SBJ}$, (right) $C\text{-}LUR_{ct_2,SBJ}$

the order of application affects the final result.

To exert an ordering on the *LURs* of different context types, we define *ContextSet* as an ordered set of context types. So, if for the previous example, $ContextSet = \langle ..., ct_1, ..., ct_2, ... \rangle$, ct_1 has higher priority to ct_2 in this respect; and therefore, $C\text{-}LUR_{ct_1,SBJ}$ is applied before $C\text{-}LUR_{ct_2,SBJ}$.

There is no general method to specify how system designer must arrange context types. But, one method would be to arrange context types based on their changing rate; i.e., the less the changing rate, the higher the priority. For instance, assume a system includes two context types *Age* and *Location* and the changing rate of *Age* is less than *Location*, then its *ContextSet* can be defined as $ContextSet = \langle Age, Location \rangle$.

An Algorithm for Updating Levels of an Entity: *UpdateEntityLevels* updates the confidentiality and integrity levels of a specific entity (which is passed to it as an argument) based on the appropriate *LURs* of all context types in *ContextSet*. Algorithm 3 is employed for this purpose.

Algorithm 3 UpdateEntityLevels($e \in EntitySet \setminus \{environment\}, ET \in \{USR, SBJ, OBJ\}$)

```
1: for all context type ct ∈ ContextSet in order do
2:   if C-LUR_{ct,e} ∈ C-LURSet_{ct} then
3:     Apply-CLUR(ct, C-LUR_{ct,e}, e)
4:   else if C-LUR_{ct,ET} ∈ C-LURSet_{ct} then
5:     Apply-CLUR(ct, C-LUR_{ct,ET}, e)
6:   end if
7:   if I-LUR_{ct,e} ∈ I-LURSet_{ct} then
8:     Apply I-LUR(ct, I-LUR_{ct,e}, e)
9:   else if I-LUR_{ct,ET} ∈ I-LURSet_{ct} then
10:    Apply-ILUR(ct, I-LUR_{ct,ET}, e)
11:  end if
12: end for
```

In this algorithm, *ET* represents the type of entity. *USR*, *SBJ*, and *OBJ* represent *User*, *Subject*, and *Object* sets respectively. The *LURs* of context types are applied based on the ordering defined by *ContextSet*. For each context type *ct*, the algorithm first checks if there is a specific *C-LUR* defined for the 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-LURs*.

The Time of *LUR* Application: If every change in context predicates triggers corresponding *LURs*, the overhead would be too high. Therefore, *LURs* must be triggered on special occasions. In fact, at two times, the confidentiality and integrity level of entities must be updated. The first time is when a user creates or activates a subject. Since the confidentiality and integrity level of a subject must be dominated by the corresponding levels of its user, the user's level must be updated before creation/activation.

The second time is before authorizing a request; i.e., when constraints must be evaluated, since the accurate value of the confidentiality and integrity level of entities are needed prior to evaluation. More details on this issue are presented in section 4.5.

4.4 Operations

An Operation can be executed by a subject on an object under specific circumstances; and, may change the state of the subject and object. *OperationSet* is the set of operations defined in the system. Formally, an operation $opr \in OperationSet$ is defined as pair $opr = \langle AccessRightSet_{opr}, Constraint_{opr} \rangle$, where

- $AccessRightSet_{opr}$ is a subset of the set $\{read, write\}$ which denotes the access right set of the operation *opr*. Note that the set of access rights in CAMAC is comprised of *read* and *write*. In CAMAC, every operation, based on what it carries out, includes a subset of these access modes; e.g., if it only does an observation of information and no alteration, it only includes read and so on.
- $Constraint_{opr}$ is composed of condition blocks for the operation *opr*. Each operation includes a constraint which denotes the prerequisite conditions that must be satisfied before the operation is executed.

There are 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 the variable *USR*, *SBJ*, and *OBJ* to represent user, subject, and object respectively. Use of these variables allows us to define generic constraints.

- 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 $op \in DomOperatorSet$. For instance, $\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle$ is a *C-CB* denoting the simple security property of BLP.
- Integral Condition Block (*I-CB*): An inte-

gral condition block is defined as a triple $\langle \omega_1, op, \omega_2 \rangle$, in which $\omega_1, \omega_2 \in IntegLvl$ and $op \in DomOperatorSet$. 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 $Value_1, Value_2 \in ValueSet_{ct}\{.element\}$, $ct \in ContextSet$, and $op \in OperatorSet$. The subscript ct determines that the *Operator-Definer* functions of the context type ct are used to evaluate the *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:

$$Constraint \rightarrow Constraint \vee C_1$$
$$Constraint \rightarrow C_1$$
$$C_1 \rightarrow C_1 \wedge C_2$$
$$C_1 \rightarrow C_2$$
$$C_2 \rightarrow (Constraint)$$
$$C_2 \rightarrow Cxt\text{-}CB | C\text{-}CB | I\text{-}CB$$

For example, for operation *GenerateReport*, the following constraint may be defined using the above grammar:
$Constraint_{GenerateReport} = (\langle \lambda(SBJ), \geq, S \rangle) \vee (\langle \lambda(SBJ), =, C \rangle \wedge \langle Time[environment][Is], \geq, 6 \rangle_{Time} \wedge \langle Time[environment][Is], \leq, 12 \rangle_{Time})$

4.5 Authorization of Users' Requests

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 $s \in Subject$, $o \in Object$, and $opr \in Operation$. Furthermore, the user of an action is the user that the subject is acting on behalf of; i.e., $u = RepOf(s)$. Algorithm 4 handles the authorization of actions.

Upon occurrence of an action, initially the confidentiality and integrity levels of user, subject, and object of the action must be updated using the *UpdateEntityLevels* algorithm. Since the confidentiality and integrity levels of a subject must be dominated by the corresponding levels of its user, after updating levels of the user and the subject, the following assignments seem indispensable:
$\lambda(s) = GLB(\lambda(s), \lambda(u))$
$\omega(s) = GLB(\omega(s), \omega(u))$

After updating the levels, the constraint of the action must be evaluated. The 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 the action based on the access right set of the operation. In other words, if $read \in AccessRightSet_{opr}$, the simple security property of BLP and simple integrity property of Biba must be added to $Constraint_A$; i.e., $Constraint_A = Constraint_A \wedge (\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle \wedge \langle \omega(OBJ), \geq, \omega(SBJ) \rangle)$.

Also, if $write \in AccessRightSet_{opr}$, the *-property of BLP and integrity *-property of Biba must be added to $Constraint_A$; i.e., $Constraint_A = Constraint_A \wedge (\langle \lambda(OBJ), \geq, \lambda(SBJ) \rangle \wedge \langle \omega(SBJ), \geq, \omega(OBJ) \rangle)$.

Finally, 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; otherwise, it is denied.

Algorithm 4 AuthorizeAction($A = \langle s, o, opr \rangle$)

1: $u = RepOf(s)$
2: $Constraint_A = Constraint_{opr}$
3: $UpdateEntityLevels(u, USR)$
4: $UpdateEntityLevels(s, SBJ)$
5: $UpdateEntityLevels(o, OBJ)$
6: $\lambda(s) = GLB(\lambda(s), \lambda(u))$
7: $\omega(s) = GLB(\omega(s), \omega(u))$
8: **if** $read \in AccessRightSet_{opr}$ **then**
9: $Constraint_A = Constraint_A \wedge (\langle \lambda(SBJ), \geq, \lambda(OBJ) \rangle \wedge \langle \omega(OBJ), \geq, \omega(SBJ) \rangle)$
10: **end if**
11: **if** $write \in AccessRightSet_{opr}$ **then**
12: $Constraint_A = Constraint_A \wedge (\langle \lambda(OBJ), \geq, \lambda(SBJ) \rangle \wedge \langle \omega(SBJ), \geq, \omega(OBJ) \rangle)$
13: **end if**
14: assign u, s, o to USR, SBJ, OBJ in $Constraint_A$ respectively
15: **return** $Evaluate(Constraint_{opr})$

5 Case Study

Mandatory models have been primarily used in multilevel security environments in which classification of information is an inherent characteristic. The following case study introduces a military environment, and uses CAMAC to satisfy its security requirements. In a military system, the entities are confidentially categorized as *TopSecret*, *Secret*, *Confidential*, and *Unclassified*. With respect to integrity, entities are categorized as *Crucial*, *VeryImportant*, and *Important*. Two kinds of read operation are defined: a normal read operation which reads *Unclassified* and *Confidential* documents; and, a military read operation which reads *Secret* and *TopSecret* ones.

The only restrictions to normal read, beyond BLP and Biba properties, is that the confidentiality level of the object must be less than or equal to $Confidential$. On the other hand, the military read requires several prerequisites:

- The read operation must be done locally; i.e., the location of subject and object must be equal.
- The read operation must be done only between 8 and 13.
- The confidentiality level of subject must be higher than or equal to S.

Moreover, there are other rules in the system such as:

- The confidentiality level of objects decreases every 10 years with a maximum of two level decrease.
- Before any operation can take place, the subject and object must be in reliable locations and the location of the user must be similar to the location of the subject.

If every location is associated with a confidentiality level based on its significance and security, then a reliable location for an entity is a place which has a confidentiality level higher than or equal to the confidentiality level of that entity [21]. The definition can be extended to take integrity into consideration too.

5.1 System Definition

We utilize CAMAC to design a system which satisfies the requirements of the environment. The system is designated as $MilitarySystem$ and is formally defined below.

$MilitarySystem = \langle EntitySet, RepOf, OperatorSet, ConfLvl, IntegLvl, \lambda, \omega, ContextPredicateSet, ContextSet, OperationSet\rangle$

- $EntitySet = \{User, Subject, Object, Environment\}$
  - $User = \{Stephan, David\}$
  - $Subject = \{Stephan\text{-}Proc, David\text{-}Proc\}$
  - $Object = \{MilitaryDoc, OfficeDoc\}$
- $RepOf(Stephan\text{-}Proc) = Stephan$, $RepOf(David\text{-}Proc) = David$
- $OperatorSet = \{>, \geq, <, \leq, =, \neq, \subset, \subseteq, \supset, \supseteq\}$
- $ConfLvl = \langle TS, S, C, U\rangle$
- $IntegLvl = \langle C, VI, I\rangle$
- $\lambda(Stephan) = TS$, $\lambda(David) = S$, $\lambda(Stephan\text{-}Proc) = TS$, $\lambda(David\text{-}Proc) = C$, $\lambda(MilitaryDoc) = TS$, $\lambda(OfficeDoc) = U$
- $\omega(Stephan) = C$, $\omega(David) = VI$, $\omega(Stephan\text{-}Proc) = C$, $\omega(David\text{-}Proc) = VI$, $\omega(MilitaryDoc) = C$, $\omega(OfficeDoc) = I$
- $ContextPredicateSet$
  - $\langle Stephan, Location, Is, HeadOffice\rangle$
  - $\langle David, Location, Is, GuestRoom\rangle$
  - $\langle David\text{-}Proc, Location, Is, GuestRoom\rangle$
  - $\langle Stephan\text{-}Proc, Location, Is, HeadOffice\rangle$
  - $\langle MilitaryDoc, Location, Is, HeadOffice\rangle$
  - $\langle OfficeDoc, Location, Is, GuestRoom\rangle$
  - $\langle MilitaryDoc, Age, Is, 27\rangle$
  - $\langle OfficeDoc, Age, Is, 11\rangle$
  - $\langle environment, Time, Is, 9\rangle$
  - $\langle HeadOffice, LocationLvl, Is, TS\rangle$
- $ContextSet = \langle Age, Location, Time\rangle$. The detailed definition of context types is given in Section 5.2.
- $OperationSet = \{NormalRead, MilitaryRead\}$. The detailed definition of operations is given in section 5.3.

5.2 Definition of Context Types

The $ContextSet$ of $MilitarySystem$ includes three context types which are formally described as follows.

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

- $ValueSet_{Time} = \{n : N | 0 \leq n \leq 24\}$
- $OperatorDefinerSet_{Time}$ is added as an external module
- $RelatorSet_{Time} = \{Is\}$
- $EntityTypeSet_{Time} = \{Environment\}$
- $LURSet_{Time} = \{C\text{-}LURSet_{Time}, I\text{-}LURSet_{Time}\}$
  - $C\text{-}LURSet_{Time} = \emptyset$
  - $I\text{-}LURSet_{Time} = \emptyset$

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

- $ValueSet_{Age} = \{n : \mathbb{N} | n \geq 0\}$
- $OperatorDefinerSet_{Age}$ is added as an external module
- $RelatorSet_{Age} = \{Is\}$
- $EntityTypeSet_{Age} = \{User, Subject, Object\}$
- $LURSet_{Age} = \{C\text{-}LURSet_{Age}, I\text{-}LURSet_{Age}\}$
  - $C\text{-}LURSet_{Age} = \{C\text{-}LUR_{Age,OBJ} =$ Figure 3$\}$
  - $I\text{-}LURSet_{Age} = \emptyset$

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

- $ValueSet_{Location} = \{HeadOffice, GuestRoom, Basement, ...\}$
- $OperatorDefinerSet_{Location}$
  $\{$
  $Operator\text{-}Definer_{Location}(A \in ValueSet_{Location}, o \in OperatorSet, B \in ValueSet_{Location})$
  $\{$
  $(A = HeadOffice \wedge B = Basement \wedge o = '\subseteq')$

...
}
}
- $RelatorSet_{Location} = \{Is, Entering\}$
- $EntityTypeSet_{Location} = \{User, Subject, Object\}$
- $LURSet_{Location} = \{C\text{-}LURSet_{Location}, I\text{-}LURSet_{Location}\}$
  - $C\text{-}LURSet_{Location} = \emptyset$
  - $I\text{-}LURSet_{Location} = \emptyset$

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

- $ValueSet_{LocationLvl} = \{TS, S, C, U\}$
- $OperatorDefinerSet_{LocationLvl}$
  {
  $Operator\text{-}Definer_{LocationLvl}($
  $A \in ValueSet_{LocationLvl}$,
  $o \in OperatorSet, B \in ValueSet_{LocationLvl})$
  {
  $(A = TS \wedge B = S \wedge o = '\geq')$
  ...
  }
  }
- $RelatorSet_{LocationLvl} = \{Is\}$
- $EntityTypeSet_{LocationLvl} = \{ValueSet_{Location}\}$
- $LURSet_{LocationLvl} = \{C\text{-}LURSet_{LocationLvl}, I\text{-}LURSet_{LocationLvl}\}$
  - $C\text{-}LURSet_{LocationLvl} = \emptyset$
  - $I\text{-}LURSet_{LocationLvl} = \emptyset$

5.3 Definition of Operations

The *OperationSet* of *MilitarySystem* includes two operations: *NormalRead* and *MilitaryRead*. These operations are formally defined here.

- $NormalRead = \langle AccessRightSet_{NormalRead}, Constraint_{NormalRead}\rangle$
  - $AccessRightSet_{NormalRead} = \{read\}$
  - $Constraint_{NormalRead} = \langle\lambda(OBJ), \leq, C\rangle \wedge \langle LocationLvl[Location[SBJ][Is]][Is], \geq, \lambda(SBJ)\rangle_{LocationLvl} \wedge \langle LocationLvl[Location[OBJ][Is]][Is], \geq, \lambda(OBJ)\rangle_{LocationLvl}$
- $MilitaryRead = \langle AccessRightSet_{MilitaryRead}, Constraint_{MilitaryRead}\rangle$
  - $AccessRightSet_{MilitaryRead} = \{read\}$
  - $Constraint_{MilitaryRead} = \langle\lambda(SBJ), \geq, S\rangle \wedge \langle Time[environment][Is], \geq, 8\rangle_{Time} \wedge \langle Time[environment][Is], \leq, 13\rangle_{Time} \wedge \langle Location[SBJ][Is], =, Location[OBJ][Is]\rangle_{Location} \wedge \langle LocationLvl[Location[SBJ][Is]][Is], \geq, \lambda(SBJ)\rangle_{LocationLvl} \wedge \langle LocationLvl[Location[OBJ][Is]][Is], \geq, \lambda(OBJ)\rangle_{LocationLvl}$

5.4 Handling Actions

By defining the operations, the specification of *MilitarySystem* was finalized. Now let's consider some exemplary actions and the consequence of the *AuthorizeAction* algorithm on them.

- $A : \langle David\text{-}Proc, NormalRead, MilitaryDoc\rangle$
  Since $Age[MilitaryDoc][Is] = 27$ and based on $C\text{-}LUR_{Age,OBJ}$, the confidentiality level of *MilitaryDoc* decreases from TS to S. Notice that the $C\text{-}LURSet_{Location}$ and $C\text{-}LURSet_{Time}$ are empty. Therefore, $\lambda(MilitaryDoc) = S$ and $\lambda_{Age}(MilitaryDoc) = TS$. Now, the constraint must be evaluated:
  $Constraint_A = \langle\lambda(MilitaryDoc), \leq, C\rangle \wedge \langle LocationLvl[Location[David\text{-}Proc][Is]][Is], \geq, \lambda(David\text{-}Proc)\rangle_{LocationLvl} \wedge \langle LocationLvl[Location[MilitaryDoc][Is]][Is], \geq, \lambda(MilitaryDoc)\rangle_{LocationLvl}$
  The constraint fails because $(\langle\lambda(MilitaryDoc), \leq, C\rangle)$ is not evaluated to true.
- $B : \langle Stephan\text{-}Proc, MilitaryRead, MilitaryDoc\rangle$
  Since $Age[MilitaryDoc][Is] = 27$ and based on $C\text{-}LUR_{Age,OBJ}$, the confidentiality level of *MilitaryDoc* is decreased from TS to S. Therefore, $\lambda(MilitaryDoc) = S$ and $\lambda_{Age}(MilitaryDoc) = TS$. Now, the constraint must be evaluated.
  $Constraint_B = \langle\lambda(Stephan\text{-}Proc), \geq, S\rangle \wedge \langle Time[environment][Is], \geq, 8\rangle_{Time} \wedge \langle Time[environment][Is], \leq, 13\rangle_{Time} \wedge \langle Location[Stephan\text{-}Proc][Is], =, Location[MilitaryDoc][Is]\rangle_{Location} \wedge \langle LocationLvl[Location[Stephan\text{-}Proc][Is]][Is]), \geq, \lambda(Stephan\text{-}Proc)\rangle_{LocationLvl} \wedge \langle LocationLvl[Location[MilitaryDoc][Is]][Is]], \geq, \lambda(MilitaryDoc)\rangle_{LocationLvl}$
  The above constraint is evaluated to true since all the conditions are satisfied.

6 CAMAC Expressiveness

In this section, we explore CAMAC in terms of expressiveness, with which various mandatory concepts are defined. The expression of the Dion model has also been done, but is omitted here due to the lack of space and can be accessed in http://nsc.sharif.edu/OnlineAppendixes/camac.htm.

6.1 Set of Categories

As discussed in section 2.1, the confidentiality levels in the original BLP model are defined by two com-

ponents: a classification and a set of categories. On the other hand, as defined in section 4.1, the confidentiality levels of CAMAC consist of the first component; the set of categories is simply ignored. The same statement holds for the 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 the considered environment. They refer to the application area to which information pertains or where data is to be used. A classic example of such 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:

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

- $ValueSet_{C\text{-}Category} = \mathcal{P}(\{Nato, Nuclear, Crypto\})$
- $OperatorDefinerSet_{C\text{-}Category}$
  {
  $Operator\text{-}Definer_{C\text{-}Category}(A \in ValueSet_{C\text{-}Category}, o \in OperatorSet, B \in ValueSet_{C\text{-}Category})$
  {
  $(A = \{Nato\} \wedge B = \{Nato, NuClear\} \wedge o =' \subseteq')$
  ...
  }
  }
- $RelatorSet_{C\text{-}Category} = \{Is\}$
- $EntityTypeSet_{C\text{-}Category} = \{User, Subject, Object\}$
- $LURSet_{C\text{-}Category} = \{C\text{-}LURSet_{C\text{-}Category}, I\text{-}LURSet_{C\text{-}Category}\}$
  - $C\text{-}LURSet_{C\text{-}Category} = \emptyset$
  - $I\text{-}LURSet_{C\text{-}Category} = \emptyset$

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

for all $opr \in OperationSet$ **do**
  if $read \in AccessRightSet_{opr}$ **then**
    $Constraint_{opr} = (Constraint_{opr}) \wedge (\langle C\text{-}Category[OBJ][Is], \subseteq, C\text{-}Category[SBJ][Is]\rangle_{C\text{-}Category})$
  end if
  if $write \in AccessRightSet_{opr}$ **then**
    $Constraint_{opr} = (Constraint_{opr}) \wedge (\langle C\text{-}Category[SBJ][Is], \subseteq, C\text{-}Category[OBJ][Is]\rangle_{C\text{-}Category})$
  end if
end for

Assume that:
$opr \in OperationSet$ and $read \in AccessRightSet_{opr}$. The confidentiality level $L_1 = (c_1, s_1)$ is higher or equal to (dominates) the level $L_2 = (c_2, s_2)$ if and only if $c_1 \geq c_2$ and $s_1 \supseteq s_2$.

Notice that an action $A = \langle s, o, opr\rangle$ is authorized if the following condition blocks are true:

$\langle \lambda(s), \geq, \lambda(o)\rangle$
$\langle C\text{-}Category[o][Is], \subseteq, C\text{-}Category[s][Is]\rangle$

These condition blocks denote the first and second relationships respectively. Since both of these conditions must be satisfied for an action including *opr* to be authorized, it has the same effect as incorporating a set of categories into confidentiality levels.

6.2 Chinese Wall Policy

The Chinese Wall policy proposed by Brewer and Nash [2] arises from that segment of the commercial sector that provides consulting services to other companies. Consultants deal with confidential company information for their clients. However, a consultant should not have access to information about, say, two banks or two oil companies because such information creates a conflict of interest in the consultant's analysis and is a disservice to clients. To incorporate Chinese Wall policy into CAMAC, a lattice-based model for enforcing this policy, which was introduced by Sandhu [30], is used. In this model, company information is categorized into mutually disjoint conflict-of-interest classes. The Chinese Wall policy requires that a consultant not be able to read information for more than one company in any given conflict-of-interest class. This policy applies uniformly to users and subjects.

The policy for writing public or company information is derived from its consequence on providing possible indirect read access contrary to mandatory read controls. In this respect, users and subjects (possibly infected with Trojan horses) must be treated differently. According to [31], the policy for writing is essentially the same as the BLP *-property. To make this statement meaningful, the model defines a lattice of labels in a way that the BLP properties can be effectively used to enforce the policy.

Assume there are n conflict-of-interest classes: COI_1, COI_2,... ,COI_n each with m_i companies, so that $COI_i = \{1, 2, ..., m_i\}$, for $i = 1, 2, ..., n$. Each object in the system is labeled with the companies from which it contains information. Thus, an object that contains information from the bank A and the oil company OC is labeled $\{bank\ A,\ oil\ company\ OC\}$. Assume that banks and oil companies are distinct

conflict-of-interest classes. Then, labels such as $\{bank\ A,\ bank\ B,\ oil\ company\ OC\}$ are clearly contrary to the Chinese Wall policy.

Each label is defined as an n-element vector $[i_1, ..., i_n]$, where either $i_k \in COI_k$ or $i_k = \perp$ for $k = 1...n$. An object labeled $[i_1, ..., i_n]$ is interpreted as (possibly) containing information from company i_1 of COI_1, company i_2 of COI_2, and so on. When an element of the label is $\perp$ rather than a number, the object can not have information from any company in the corresponding COI class.

A newly enrolled user in the system is assigned the clearance $[\perp, \perp, ..., \perp]$. As the user reads company information, his/her label is updated. For example, by reading information about company 1 in COI_1, his/her label changes to $[1, \perp, ..., \perp]$. The floating up of a user's clearance corresponds with the ability to create subjects with new labels for that user. Each subject has a fixed label dominated by the label of its user at the time (s)he logged in.

The dominance relation among labels is defined as $l_1 \geq l_2$ provided l_1 and l_2 agree wherever $l_2 \neq \perp$. For example, $[1, 3, 2] \geq [1, 3, \perp]$ while $[1, 3, 2]$ and $[1, 2, 3]$ are incomparable. Based on these labels, the BLP simple security property and *-property can be exploited to enforce the Chinese Wall policy.

In this respect, based on the previous assumptions, a context type CWP is defined as follows:
$CWP = \langle ValueSet_{CWP}, OperatorDefinerSet_{CWP}, RelatorSet_{CWP}, EntityTypeSet_{CWP}, LURSet_{CWP} \rangle$

- $ValueSet_{CWP}.m_1 = (COI_1 \cup \{\perp\})$
  $ValueSet_{CWP}.m_2 = (COI_2 \cup \{\perp\})$
  ...
  $ValueSet_{CWP}.m_n = (COI_n \cup \{\perp\})$
- $OperatorDefinerSet_{CWP}$
  {
  $Operator\text{-}Definer_{CWP}(A \in ValueSet_{CWP}, o \in OperatorSet, B \in ValueSet_{CWP})$
  {
  $(o =' \geq' \wedge (B.m_1 = A.m_1 \vee B.m_1 = \perp) \wedge ... \wedge (B.m_n = A.m_n \vee B.m_n = \perp))$
  $\vee$
  ...
  }
  }
- $RelatorSet_{CWP} = \{Is\}$
- $EntityTypeSet_{CWP} = \{User, Subject, Object\}$
- $LURSet_{CWP} = \{C\text{-}LURSet_{CWP}, I\text{-}LURSet_{CWP}\}$
  - $C\text{-}LURSet_{CWP} = \emptyset$
  - $I\text{-}LURSet_{CWP} = \emptyset$

Constraints of all operations in $OperationSet$ must be changed in the following manner:

for all $opr \in OperationSet$ **do**
  if $read \in AccessRightSet_{opr}$ **then**
    $Constraint_{opr} = (Constraint_{opr}) \wedge (\langle CWP[SBJ][Is], \geq, CWP[OBJ][Is] \rangle_{CWP})$
  end if
  if $write \in AccessRightSet_{opr}$ **then**
    $Constraint_{opr} = (Constraint_{opr}) \wedge (\langle CWP[OBJ][Is], \geq, CWP[SBJ][Is] \rangle_{CWP})$
  end if
end for

It is evident that augmenting constraints with such condition blocks incorporates the Chinese Wall policy into the model.

6.3 Location-Based Mandatory Access Control Model

The location-based mandatory access control model presented by Ray *et al.* [21] is an extension of the BLP model. In this model, a confidentiality level is assigned to every location; and, BLP's simple security property and *-property are extended with the notion of location and its confidentiality level.

Specifically, at first a set of locations is defined and the relations $=$ and $\subseteq$ are determined on this set. $Loc_A \subseteq Loc_B$ denotes that Loc_A is included in Loc_B; and, $Loc_A = Loc_B$ means that both relationships $Loc_A \subseteq Loc_B$ and $Loc_B \subseteq Loc_A$ hold.

Also, a confidentiality level is assigned to every location. An entity is in a reliable location if the confidentiality level of that location dominates the confidentiality level of the entity.

Accordingly, a subject s, the representative of a user u, can read an object o (simple security property) if:
$\lambda(s) \geq \lambda(o) \wedge Loc_s \subseteq Loc_{read\_sub} \wedge Loc_o \subseteq Loc_{read\_obj} \wedge \lambda(s) \leq \lambda(Loc_{read\_sub}) \wedge \lambda(o) \leq \lambda(Loc_{read\_obj}) \wedge Loc_u = Loc_s$
Also, a subject s, the representative of a user u, can write in an object o (*-property) if:
$\lambda(o) = \lambda(s) \wedge Loc_s \subseteq Loc_{write\_sub} \wedge Loc_o \subseteq Loc_{write\_obj} \wedge \lambda(s) \leq \lambda(Loc_{write\_sub}) \wedge \lambda(o) \leq \lambda(Loc_{write\_obj}) \wedge Loc_u = Loc_s$

In these relations, Loc_u, Loc_s, and Loc_o represent the location of the user, the subject, and the object respectively. Also, Loc_{read_sub} and Loc_{read_obj} denote the permissible location of the subject and the object before a read operation can take place. Loc_{write_sub} and Loc_{write_obj} represent the same for a write operation.

Assume that $LocationSet = \{loc_1, ..., loc_n\}$. Notice that $Loc_{read\_sub}, Loc_{write\_sub}, Loc_{read\_obj}, Loc_{write\_obj} \in LocationSet$.

In order to incorporate this model into CAMAC,

at first, a context type *Location* is defined as follows.
$Location = \langle ValueSet_{Location}, OperatorDefinerSet_{Location}, RelatorSet_{Location}, EntityTypeSet_{Location}, LURSet_{Location} \rangle$

- $ValueSet_{Location} = LocationSet = \{loc_1, ..., loc_n\}$
- $OperatorDefinerSet_{Location}$ {
  $Operator\text{-}Definer_{Location}(A \in ValueSet_{Location}, o \in OperatorSet, B \in ValueSet_{Location})$
  {
  $(A = loc_i \wedge B = loc_j \wedge o = '\subseteq')$
  $\vee$
  ...
  }
  }
- $RelatorSet_{Location} = \{Is\}$
- $EntityTypeSet_{Location} = \{User, Subject, Object\}$
- $LURSet_{Location} = \{C\text{-}LURSet_{Location}, I\text{-}LURSet_{Location}\}$
  - $C\text{-}LURSet_{Location} = \emptyset$
  - $I\text{-}LURSet_{Location} = \emptyset$

Notice that $OperatorDefinerSet_{Location}$ defines the operator $\subseteq$ on *LocationSet*. Furthermore, the context type *LocationLvl* is defined. This context type simply assigns a confidentiality level to every location in *LocationSet*.
$LocationLvl = \langle ValueSet_{LocationLvl}, OperatorDefinerSet_{LocationLvl}, RelatorSet_{LocationLvl}, EntityTypeSet_{LocationLvl}, LURSet_{LocationLvl} \rangle$

- $ValueSet_{LocationLvl} = ConfLvl = \{l_1, ...l_n\}$
- $OperatorDefinerSet_{LocationLvl}$ {
  $Operator\text{-}Definer_{LocationLvl}($
  $A \in ValueSet_{LocationLvl}, o \in OperatorSet,$
  $B \in ValueSetLocationLvl)$
  {
  $(A = l_i \wedge B = l_{i+1} \wedge o = '\geq')$
  ...
  }
  }
- $RelatorSet_{LocationLvl} = \{Is\}$
- $EntityTypeSet_{LocationLvl} = \{ValueSet_{Location}\}$
- $LURSet_{LocationLvl} = \{C\text{-}LURSet_{LocationLvl}, I\text{-}LURSet_{LocationLvl}\}$
  - $C\text{-}LURSet_{LocationLvl} = \emptyset$
  - $I\text{-}LURSet_{LocationLvl} = \emptyset$

Now, we define two operations: *LocBased-Read* representing the simple security property and *LocBased-Write* representing the *-property.
$LocBased\text{-}Read = \langle AccessRightSet_{LocBased\text{-}Read}, Constraint_{LocBased\text{-}Read} \rangle$

- $AccessRightSet_{LocBased\text{-}Read} = \{read\}$
- $Constraint_{LocBased\text{-}Read} = \langle Location[SBJ][Is], \subseteq, Loc_{read\_sub} \rangle_{Location} \wedge$
  $\langle Location[OBJ][Is], \subseteq, Loc_{read\_obj} \rangle_{Location}$
  $\wedge \langle LocationLvl[Loc_{read\_sub}][Is], \geq, \lambda(SBJ) \rangle_{LocationLvl}$
  $\wedge \langle LocationLvl[Loc_{read\_obj}][Is], \geq, \lambda(OBJ) \rangle_{LocationLvl}$
  $\wedge \langle Location[SBJ][Is], =, Location[USR][Is] \rangle_{Location}$

$LocBased\text{-}Write = \langle AccessRightSet_{LocBased\text{-}Write}, Constraint_{LocBased\text{-}Write} \rangle$

- $AccessRightSet_{LocBased\text{-}Write} = \{write\}$
- $Constraint_{LocBased\text{-}Write} = \langle Location[SBJ][Is], \subseteq, Loc_{write\_sub} \rangle_{Location} \wedge$
  $\langle Location[OBJ][Is], \subseteq, Loc_{write\_obj} \rangle_{Location}$
  $\wedge \langle LocationLvl[Loc_{write\_sub}][Is], \geq, \lambda(SBJ) \rangle_{LocationLvl}$
  $\wedge \langle LocationLvl[Loc_{write\_obj}][Is], \geq, \lambda(OBJ) \rangle_{LocationLvl}$
  $\wedge \langle Location[SBJ][Is], =, Location[USR][Is] \rangle_{Location}$

It is evident that these two operations exactly have the same effect as those of the Location-based Mandatory Access Control model.

7 CAMAC as a Context-Sensitive Information Flow Control Model

Information flow control defines the way through which information moves throughout a system. Typically, information flow control policies are designed to preserve confidentiality of data or integrity of information. In the former, the policy's goal is to prevent information from flowing to an entity not authorized to receive it. In the latter, information may flow only to entities that are no more trustworthy than the data [32].

Information flow is usually controlled through assigning a security class to every entity. Whenever information flows from object x to object y, there is an accompanying information flow from the security class of x to the security class of y. Denning [33] defined the concept of an information flow policy as a triple $\langle SC, \rightarrow, \oplus \rangle$ where SC is a set of security classes, $\rightarrow \subseteq SC \times SC$ is a binary can-flow relation on SC, and $\oplus : SC \times SC$ is a class-combining or join operator on SC [33].

Based on this definition, we specify the CAMAC information policy here. A security class in the CAMAC model is defined as an ordered pair $\langle c, i \rangle$ in which $c \in ConfLvl$ and $i \in IntegLvl$. Assume that $ConfLvl = \langle \lambda_n, ..., \lambda_1 \rangle$ and $IntegLvl = \langle \omega_m, ..., \omega_1 \rangle$. Therefore, SC contains $n * m$ security classes.

Information can flow from a security class $sc_i = \langle \lambda_x, \omega_y \rangle$ to another security class $sc_j = \langle \lambda_z, \omega_w \rangle$ if
$\lambda_x \geq \lambda_z$ and $\omega_w \geq \omega_y$

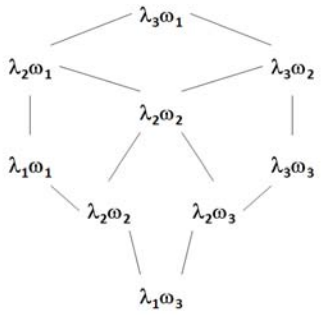

Figure 7. The Hass diagram of the CAMAC information flow policy for $m = n = 3$

Since can-flow relationship is transitive, it can be defined in the following manner:

$\langle \lambda_i, \omega_j \rangle \rightarrow \langle \lambda_{i+1}, \omega_j \rangle$ for $i : 1..n-1;\ j : 1..m$

$\langle \lambda_i, \omega_j \rangle \rightarrow \langle \lambda_i, \omega_{j-1} \rangle$ for $i : 1..n;\ j : 2..m$

Furthermore, join operator can be defined as follows:

$\langle \lambda_x, \omega_y \rangle \oplus \langle \lambda_z, \omega_w \rangle = \langle \lambda_{max(x,z)}, \omega_{min(y,w)} \rangle$

for $x, z : 1..n;\ y, w : 1..m$

Figure 7 shows the Hass diagram of this information policy for $m = n = 3$. Denning showed that under the following assumptions, an information flow policy can form a finite lattice [33]:

(1) The set of security classes SC is finite.
(2) The can-flow relation (i.e., $\rightarrow$) is a partial order on SC.
(3) SC has a lower bound with respect to $\rightarrow$.
(4) The join operator (i.e., $\oplus$) is a totally defined least upper bound operator.

The information flow policy defined above satisfies the first assumption, since the number of members of SC is finite and equal to $m*n$. It also satisfies the second assumption, since can-flow relationship is defined as a partial order on SC. Furthermore, it satisfies the third assumption, since SC has a lower bound with respect to the can-flow relationship; i.e., $\langle \lambda_1, \omega_m \rangle$. The fourth assumption is also satisfied, because $\oplus$ is defined as a least upper bound operator and it is total; i.e., for every pair of security classes, $\oplus$ defines their least upper bound. Therefore, the CAMAC information flow policy is a lattice.

Security classes assigned to entities in the CAMAC model can be updated according to context. As mentioned in section 4.3.5, $LURs$ define how such updates occur based on changes in contextual values. Therefore, contrary to traditional lattice-based access control models, security classes associated with entities are context-sensitive in CAMAC. This property allows CAMAC to be considered as an information flow control model with a context-sensitive security class association.

8 Evaluation

The CAMAC model could be evaluated and compared with other mandatory models based on various criteria; i.e., authorization time, complexity of policy description, support for context-awareness, expressiveness, and objective. The models and policies, which we consider for this purpose, are BLP, Biba, Dion, Chinese Wall, and the Location-based Mandatory Access control model presented by Ray *et al.* [21].

8.1 Authorization Time

One important metric would be the computational time needed to authorize an action. Assuming that the retrieval of an entity's confidentiality and integrity level can take place in $O(G(e))$, where e is the total number of entities in the system and $G(e)$ is a function of e, it is easy to show that the BLP, Biba, and Dion models authorize an action based on a finite number of level comparisons; therefore, their computational time for authorization of an action would be $O(G(e))$. Furthermore, time complexity of the location-based mandatory access control model is higher than the BLP model, but still less than CAMAC.

Let's compute the computational time needed to authorize an action in the CAMAC model. Our objective is to authorize the action $Act = \langle s, o, opr \rangle$ using the *AuthorizeAction* algorithm. Assume that $RepOf(s) = u$. The time needed for retrieval of a contextual value is dependant on the implementation. If c is the average number of predicates in $ContextPredicateSet$, then the retrieval time will be equal to $F(c)$ where F is a function of c. In the worst case, $F(c) = c$. The time for retrieval of a level is simply $G(e)$. In general, $F(c)$ is more costly than $G(e)$; i.e., retrieval of contextual values takes more time than retrieval of confidentiality and integrity levels.

Assume n different context types are defined in the system. *AuthorizeAction* includes the following primary steps:

- Updating the Confidentiality and Integrity Level
- Evaluating the Constraint

Confidentiality and Integrity Level Updates: In the worst case, the $LURSet$ of every context type includes a LUR applicable to u, s, and o. Assume, $ContextSet$ includes n context types. k denotes the maximum number of statements that must be considered for any transition; e.g., for $MilitarySystem$ presented in section 5, k is 3, since three statements must be evaluated for transition from the state S to the state C. Also, m denotes the maximal computational

cost of all *Operator-Definer* functions in the system. Each call of *Operator-Definer* function requires two retrievals of contextual values; and therefore, the maximum time needed to call *Operator-Definer* is $m + 2F(c)$.

Note that the computational cost for evaluations of relationships among integrity and confidentiality levels is assumed to be 1, and considering two level retrievals, the total cost would be $2G(e) + 1$.

Therefore, the evaluation of each transition in *LURs* has a maximum computational cost of $k(G(e) + m + 2F(c))$. Since the number of states is finite, and at most three *LURs* of each context type are applicable to the entities of the action, the cost of level update for all context types would be $O(n.k.m + n.k.F(c) + n.k.G(e))$.

Constraint Evaluation: In the worst case, all condition blocks are of type *Cxt-CB*; therefore, they all use *Operator-Definer* functions. Suppose the $Constraint_{opr}$ includes at most p number of *Cxt-CBs*. One *C-CB* and one *I-CB* are added to the constraint of operation before the authorization. Also, assume that parsing of constraint takes place in an efficient way, and its cost is a linear function of *CB* numbers, i.e., $O(p)$. Since each *C-CB* and *I-CB* can be evaluated in $(2G(e) + 1)$ time, and each *Cxt-CB* can be evaluated in $(m + 2F(c))$ time, the cost of constraint evaluation is $O(p.m + p.F(c) + G(e))$.

Therefore, based on the above calculations, the total computational time of *AuthorizeAction* is $O(n.k.m + n.k.F(c) + n.k.G(e) + p.m + p.F(c))$. Besides, factors p, s and k, as seen in the example of section 4.3.5 and *MilitarySystem* presented in section 5, are usually small numbers (less than 10) and n, m and $F(c)$ are the dominant factors of computational cost. As a result of this consideration, the cost is $O(nm + nF(c))$. Notice that $G(e)$ can be ignored due to $F(c)$. In other words, computational time in the CAMAC model is mostly dependant on the number of context types, number of context predicates, and maximum time complexity of all *Operator-Definer* functions.

Also notice that the number of context predicates is at most the number of entities multiplied by the number of context types; i.e. $c = n * e$. If the number of entities is polynomial in terms of the number of context types (which is true in almost all cases), $O(n.F(c))$ is polynomial. Thus, for the computational time to be polynomial, m, the maximum of time complexities of all *Operator-Definer* functions, must be polynomial. This assumption may not be necessarily true in all cases.

8.2 Other Metrics

Another important metric would be the effort needed to specify policies in the model. There is no accurate measure to evaluate the degree of complexity. Informally, although CAMAC is flexible in the specification of access control policies, it requires more efforts and complicated processes than BLP and other models.

In terms of expressiveness, as shown in section 6, many mandatory models can be specified by the CAMAC model; therefore, it is more expressive than other mandatory models and policies.

Furthermore, CAMAC, like Dion, preserves both confidentiality and integrity of information. In this respect, BLP and Chinese Wall only preserve confidentiality, while Biba only preserves integrity.

A primary objective of CAMAC is provision of context-awareness. As mentioned in section 2.2, location-based mandatory access control model can be categorized as a location-aware mandatory model.

Table 1 compares mandatory access control models with different criteria.

9 Conclusion

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 of levels and definition of more sophisticated mandatory policies. We showed that the authorization time of CAMAC has increased in comparison with the BLP and Biba models. This results from the ability of incorporating various mandatory controls into the CAMAC model. In CAMAC, BLP policy and Biba strict integrity policy are the built-in part of the model and other Biba policies, Chinese Wall policy and somehow Dion policy can be appended to the model using context types. Furthermore, combination of mandatory policies can be used simultaneously. For instance, BLP, Biba strict integrity policy and Chinese Wall policy can all be deployed at once.

Also, CAMAC can be deployed in the environments where information flow control with context-sensitive security class association is needed.

Granted increase in the authorization time and complexity of the model, integration of context-awareness into the model can enhance its flexibility in certain ways. It allows us to define more sophisticated mandatory policies using contextual information. Moreover, context-sensitivity of confidentiality and integrity levels, enables the model to directly

Table 1. Comparison of different MAC models

	CAMAC	BLP	Biba	Dion	Chinese Wall	LMAC
Policy Specification Complexity	high	low	low	medium	medium	medium
Preserves Confidentiality and Integrity	both	confidentiality	integrity	both	confidentiality	confidentiality
Supports Context-Awareness	yes	no	no	no	no	only locations
Expressiveness	high	low	low	medium	low	medium
Computational Cost	high	low	low	low	medium	medium

address a group of security requirements of multi-level security environments which have been simply overlooked in other context-aware models.

References

[1] Ravi S. Sandhu and Pierangela Samarati. Access Controls: Principles and Practice. *IEEE Communications*, 32:40–48, 1994.

[2] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium Research in Security and Privacy*, pages 215–228, Los Alamitos, CA, 1989. IEEE CS Press.

[3] DoD. *US Department of Defense Trusted Computer System Evaluation Criteria (The Orange Book)*. US Department of Defence, Washington DC, 1987.

[4] Silvana Castano, Maria Grazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. Addison-Wesley and ACM Press, 1995.

[5] David E. Bell and Leonard J. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical report, MITRE Corporation, 1976.

[6] David E. Bell and Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical report, MITRE Corporation, 1976.

[7] K. Biba. Integrity Considerations for Secure Computer Systems. Technical report, 1977.

[8] L. C. Dion. A Complete Protection Model. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 49–55, Oakland, CA, 1981.

[9] Bill N. Schilit, Norman I. Adams, and Roy Want. Context-aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, 1994. IEEE Computer Society.

[10] Matthias Baldauf and Schahram Dustdar. A Survey on Context-Aware Systems. Technical report, Distributed Systems Group , Technical University of Vienna, 2004.

[11] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *Proceedings of the Second International Symposium on Wearable Computers*, pages 92–99, Pittsburgh, PA, USA, 1998. EEE Computer Society Press.

[12] Anind K. Dey. Context-Aware Computing: The CyberDesk Project. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, pages 51–54, Menlo Park, CA, 1998. Technical Report, SS-98-02.

[13] Anind K. Dey and Gregory D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99)*, pages 304–307, London, UK, 2000. Springer-Verlag.

[14] Mari Korkea-Aho. Context-Aware Applications Survey, 2000.

[15] Arun Kumar, Neeran Karnik, and Girish Chafle. Context Sensitivity in Role Based Access Control. *Proceedings of the ACM SIGOPS Operating Systems Review*, pages 53–66, 2002.

[16] Mohammad A. Al-Kahtani and Ravi Sandhu. A Model for Attribute-Based User-Role Assignment. In *Proceedings of the 18th Annual Computer Security Applications Conference*, pages 353–364, Las Vegas, NV, USA, 2002. IEEE Computer Society Press.

[17] M. Covington, M. Moyer, and M. Ahamad. Generalized Role-Based Access Control for Securing Future Applications. In *Proceedings of the 23rd National Information Systems Security Conference*, Baltimore, MD, USA, 2000.

[18] G. Zhang and M. Parashar. Context-Aware Dynamic Access Control for Pervasive Applications. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 219–225, San Diego, CA, USA, 2004.

[19] C.K. Georgiadis, I. Mavridis, G. Pangalos, and R.K. Thomas. Flexible Team-based Access Control Using Contexts. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, pages 21–27, Chantilly, VA, USA, 2001. ACM Press.

[20] J. Hu and A. C. Weaver. A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications. In *Proceedings of the First Workshop on Pervasive Privacy Security, Privacy, and Trust*, Boston, MA, USA, 2004.

[21] Indrakshi Ray and Mahendra Kumar. Towards a Location-Based Mandatory Access Control Model. *Computers & Security*, 25:36–44, 2006.

[22] Manuel Roma'n, Christopher Hess, Renato Cerqueira, and Anand Ranganathan. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[23] Amir Reza Masoumzadeh, Morteza Amini, and Rasool Jalili. Context-Aware Provisional Access Control. In *Proceedings of the Second International Conference On Information Systems Security*, volume 4332, pages 132–146, Kolkata, India, 2006. Published in Lecture Notes in Computer Science.

[24] Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Kernen, and Esko-Juhani Malm. Managing Context Information in Mobile Devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003.

[25] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. A Middleware for Building Context-Aware Mobile

Services. In *Proceedings of the IEEE Vehicular Technology Conference*, volume 5, pages 2656–2660, Milan, Italy, 2004.

[26] Patrick Fahy and Siobhan Clarke. CASS: Middleware for Mobile, Context-Aware Applications. In *Proceedings of the Workshop on Context Awareness at MobiSys*, pages 304–308, Boston, 2004.

[27] Harry Chen, Tim Finn, and Anupam Joshi. Using OWL in a Pervasive Computing Broker. In *Proceedings of the Workshop on Ontologies in Open Agent Systems (AAMAS'03)*, pages 9–16, Melbourne, Australia, 2003.

[28] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. 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.

[29] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. In *Proceedings of the 36th Hawaii International Conference on System Sciences*, Hawaii, USA, 2003.

[30] Ravi S. Sandhu. A Lattice Interpretation of the Chinese Wall Policy. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 329–339, Washington, D.C., 1992. US Government Printing Office.

[31] Ravi S. Sandhu. Lattice-Based Access Control Models. *IEEE Computer*, 26(11):9–19, 1993.

[32] Mat Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[33] D. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.

Jafar Haadi Jafarian received the BS degree in software engineering from University of Tehran, Tehran, Iran, in 2003 and the MS degree in IT from Sharif University of Technology, Tehran, Iran, in 2007. He is currently the manager of a CERT group in Sharif Network Security Center (security research lab of Sharif University of Technology), Tehran, Iran. His research interest are access control, privacy and anonymity, and vulnerability analysis.

Morteza Amini received the BS degree in software engineering from Shahid Beheshti University, Tehran, Iran, in 2001 and the MS degree in software engineering from Sharif University of Technology, Tehran, Iran, in 2004. He is currently a PhD candidate in software engineering in Sharif University of Technology. His research interests include computer security, access control, intrusion detection systems, and e-banking security. He also handles access control research group in Sharif Network Security Center (security research lab of Sharif University of Technology), Tehran, Iran.