The ISC Int'l Journal of Information Security

January 2010, Volume 2, Number 1 (pp. 47–66)

http://www.isecure-journal.org

A Context-Sensitive Dynamic Role-Based Access Control Model for Pervasive Computing Environments

Sareh Sadat Emami [a,*], Saadan Zokaei [b]

[a] Electrical Engineering Department, K. N. Toosi University of Technology, Tehran, Iran and Network Security Center (NSC), Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

[b] Electrical Engineering Department, K. N. Toosi University of Technology, Tehran, Iran

ARTICLE INFO.

Article history:

Received: 9 March 2009

Revised: 28 September 2009

Accepted: 13 December 2009

Published Online: 26 January 2010

Keywords:

Access Control, Pervasive Computing Environment, Long-Term Context, Short-Term Context, Dynamic Role-Assignment, Dynamic Permission-Activation

ABSTRACT

Resources and services are accessible in pervasive computing environments from anywhere and at anytime. Also, due to ever-changing nature of such environments, the identity of users is unknown. However, users must be able to access the required resources based on their contexts. These and other similar complexities necessitate dynamic and context-aware access control models for such environments. In other words, an efficient access control model for pervasive computing environments should be aware of context information. Changes in context information imply some changes in the users' authorities. Accordingly, an access control model for a pervasive computing environment should control all accesses of unknown users to the resources based upon the participating context information, i.e., contexts of the users, resources and the environment. In this paper, a new context-aware access control model is proposed for pervasive computing environments. Contexts are classified into long-term contexts (which do not change during a session) and short-term contexts (which their steady-state period is less than an average time of a session). The model assigns roles to a user dynamically at the beginning of their sessions considering the long-term contexts. However, during a session the active permission set of the assigned roles are determined based on the short-term context conditions. Formal specification of the proposed model as well as the proposed architecture are presented in this paper. Furthermore, by presenting a real case study, it is shown that the model is applicable, decidable, and dynamic. Expressiveness and complexity of the model is also evaluated.

1 Introduction

Pervasive computing environments (henceforth called PCEs) contain heterogeneous users and resources. Maybe, there exist different sorts of users and services, which are not even predefined. Nevertheless, users should be able to use authorized resources and services at any time and location [1]. A PCE commonly includes the following four elements [2]:

I. Devices: Users use these devices to communicate with the environment (such as PDAs).
II. Pervasive Network: It is an infrastructure that

* Corresponding author.

Email addresses: emami@ee.kntu.ac.ir (S. S. Emami), szokaei@eetd.kntu.ac.ir (S. Zokaei).

ISSN: 2008-2045

enables establishing connections between nodes; i.e., users' devices, resources, and service generators.

III. Middleware: It acts as a medium between the pervasive network and applications. It manages connections among the nodes of the network according to the users' requests and network rules.

IV. Applications: Users access the pervasive network using the applications. In fact, the applications act as interfaces between users and the middleware.

Since PCEs should involve the interaction of numerous, casually accessible, and often invisible computing devices, security (specifically access control) is important challenge in such environments. An access control system is incorporated into the middleware in a PCE to control access requests in the network.

Access Control is the process by which an entity such as a user gets the right required to perform an operation. Access control in a distributed environment should be determined as a result of evaluating the request of an authenticated user for accessing some resources, against various policies [3]. There are many authorization mechanisms such as access control list (ACL), role-based authorization, rule-based authorization and identity-based authorization. But these mechanisms alone cannot satisfy the access requirements of distributed environments. Because many factors affect the access control in such environments, such as privacy requirements of the requester, attributes and identities of resources and requesters, and context properties [4].

In other words, regarding mobility and heterogeneity of users in PCEs, context information and users' attributes play a crucial role in the access control process. Dey and Abowd [5] defined context as any information that can be used to characterize the situation of an entity. An entity might be a person, a place, or an object that is relevant to the interaction between a user and an application, including the user and application themselves. Context may include entities' attributes, date, time, location, system capabilities, and any other information about the entities or the environment.

Following the above discussion, context awareness is necessary for an efficient access control in PCEs. This requirement is not covered in traditional access control models [6] such as Role-Based Access Control (RBAC) models [7]. The RBAC models have drawn the attention of recent access control systems since they are easily administrable, and compatible with the organizations' structure. However, RBAC is not an appropriate model for PCEs, because applying user-role assignment in such environments is impossible due to the existence of undefined users. Role management is also hard due to the existence of numerous roles. Role and permission assignments are static, and context information is not taken into consideration in access control decision procedure [8].

In this paper, we propose a role-based access control model which is compatible with the PCE properties. Prerequisite context predicates are defined for role assignments, and dynamic user-role assignment is applied in a session according to the changes in context information. Hence, contextual preconditions are defined for activating roles' permissions.

The rest of this paper is organized as follows. Section 2 describes related work and other approaches to access control in pervasive computing environments. Our proposed access control model is presented in Section 3; this section contains formal definition of the model. A case study is demonstrated in Section 4. Section 5 introduces an architecture for the access control system. Section 6 evaluates the model and the architecture, and finally Section 7 concludes the paper.

2 Related Work

Every access control approach in pervasive computing environments uses context. Some of them (e.g., see [9, 10]) use rules for granting permissions, but considering lots of the objects and subjects in PCEs, rule-based access control is not an effective solution. Most of the proposed models for such environments are extended RBAC; they limit role-permission assignments using context, but do not assign roles to users dynamically. Thus, in these models users are supposed to be known. Also, there are lots of roles and users in these models, which their management seems impossible in practice. In this section, some proposed access control models and frameworks for PCEs are studied.

A context-sensitive access control model based on RBAC is proposed in [6]. The model has four predefined roles for context management: Context Owner (CO), Context Provider (CP), Context Broker (CB), and Context-Aware Service Provider (CASP). They focus on context information assurance and secure transmission of context among the pervasive nodes. The model needs an infrastructure for the collection, management, and interpretation of context information. This model is not a perfect model for PCEs, because the signaling overhead of periodic polling of fresh context information and its verification might be too much for many context-aware infrastructures. Furthermore, security policies and access control rules are not observed in the model, and the specification of security policies with contextual conditions is not supported.

Zhang and Parashar [11] proposed a dynamic role-based access control model that extends the RBAC model. In this model, roles are assigned to users, analogous to RBAC (static user-role assignment) and users' roles are activated dynamically based on the context changes in each session. Two state machines are defined in this model, one for activating users' roles in sessions and another for role-permission assignment for each object based on context changes. There is a central authorizer which assigns a role state machine to each user's agent and changes the active role in the state machine according to the user's contexts changes. Each object has a permission state machine, which is modified when the contexts change. It means there is one state machine for each object and one role-state machine for each user. Since there are numerous users and objects in pervasive environments, there are too many state machines in the model that it is almost impossible to generate. Hence, due to static user-role assignment and countless state machines for users, this model is not appropriate for PCEs.

Hengartner and Steenkiste [12] proposed two models for access control in PCEs; End-to-End and Step-by-Step models. In the End-to-End model, a source node validates that the sink node and every middle node are authorized to receive the requested piece of information emanating from the source node. The advantage of this model is that access control is done at a single point; there are no redundant access control checks. The drawback of the model is that, it puts a heavy load on a source node. Also, a request for information has to flow through the entire system to the source node before an access control decision is made. Intermediate nodes process the request and potentially translate it into different requests. Therefore, the End-to-End model is more prone to denial-of-service attacks. In the Step-by-Step model, for each possible pair of server/client nodes participating in the information flow, the server node validates the client node. The advantage of this model is that it distributes the access control load over multiple nodes. In addition, an invalid request can be thrown away immediately by the first node receiving the request. Thus, this method is less prone to denial-of-service attacks. The drawback of this model is that all nodes need to run access control. Hence, there might be redundant checks. However, they introduce their architecture regarding the Step-by-Step model. In this case some server nodes must check the situation for sending requests, so policies are not hidden. In other words, access control is distributed among some nodes. Receivers make decisions for accesses to themselves and they must get ensured over their requesters' authenticity.

CACM, Context-aware Access Control Model, based on $UCON_{ABC}$, is proposed in [13]. $UCON_{ABC}$ model is based on the following three criteria: 1) decision factors that consist of Authorizations, oBligations, and Conditions, 2) continuity of decision being either pre or ongoing with respect to the access in the question, 3) mutability that can allow updates on subject or object attributes at different times. CACM emphasizes on immutable attributes and defines decision factors (Authorizations, Obligations and Conditions) according to context information. Subject and object attributes as well as environmental situations trigger changes in access privileges. However, the model does not explain how to define conditions. In other words, the definition of conditions and context predicates is not clear. Moreover, due to the infinity of context types, management of context conditions is almost impossible.

Cerberus, a context-aware security scheme for smart spaces, is proposed in [10]. The Cerberus core service of Gaia (a generic computational environment that integrates physical spaces and their ubiquitous computing devices into a programmable computing and communication system [14]) aims at capturing context information as much as possible by deploying different devices and sensors, identifying entities, and reasoning automatically in order to provide an unobtrusive computer environment. Cerberus consists of four major components: 1) the security service, 2) the context infrastructure, 3) a knowledge base (that stores various security policies), and 4) an inference engine (which performs automated reasoning and enforces the security policies). It allows principals to define context-sensitive policies based on first-order logic. It expresses context information with context predicates, and single inference engine evaluates all the authorization decisions. However, the scheme has some drawbacks to be used in PCEs. For example, since the scheme is rule-based, and there are numerous rules in PCEs, rule management is time-consuming and not sufficient for real time services. Also, context fetching in Cerberus consumes a lot of time, which causes some delay in the decisions.

Shen and Hong [15] presented a context-aware role-based access control model (CGRBAC) for web services. CGRBAC is an extension of RBAC model for global services (composite services in the web services) and context. When a user calls global services in this model, he/she uses global roles. Global roles are different from traditional roles, because they hold the role information from other providers mapped to them. In CGRBAC model, the user does not select the activated role directly. Instead, the global role activation depends on the security-relevant environmental context. Local roles activation depends on their invocation of corresponding local services (atomic services) by global services. Although the model covers the PCEs regarding heterogeneity of users, it provides limited services. So, CGRBAC is not a good solution for multi-purpose environments.

Jafarian and Amini [16] proposed a context-aware mandatory access control model, called CAMAC, which preserves the confidentiality and integrity of information as specified in the traditional mandatory access control models. Moreover, it handles dynamic adaptation of access control policies to the context, and context-sensitive class association. The model is capable of being deployed in multilevel security environments and where the information flow control with context-sensitive security classes is necessary. Also, contextual information is represented using the notion of context predicates, and context types. To show high expressiveness, context information is described formally. CAMAC uses context information perfectly; nevertheless it is not convenient for PCEs. Since CAMAC is a mandatory access control model, it can be used in specific environments such as the military ones.

CAP, a context-aware access control model for PCEs, is proposed in [17]. Although the CAP model is formed based on the concepts of RBAC model, it tries to eliminate RBAC drawbacks to be used for PCEs. As an example, user-role assignment is dynamic in the CAP model. Context information affects user-role assignment and roles' permissions activation in every request. Dynamic user-role assignment in CAP makes heterogeneous user management possible, even when they are unknown. However, this model has some drawbacks as follows: 1) it needs to fetch many context values to make a decision while some of them may not be used, so it causes high overhead at the execution time, 2) it does not support role hierarchy, and 3) it uses limited combination of context conditions for assigning roles or activating permissions.

Our proposed model, named iCAP [1], is introduced for access control in PCEs. iCAP improves the CAP model and tries to solve its problems. So, it is a role-based access control model based on the RBAC concepts. iCAP defines a role hierarchy which is assigned to users dynamically at the beginning of sessions according to the current situations and context information. So it can control accesses of heterogeneous and even unknown users. The model fetches context information when they are needed for a decision and avoids gathering extra contexts, because it is too time-consuming.

[1] iCAP: **i**mproved **C**ontext-aware **A**ccess control model for **P**ervasive computing environments

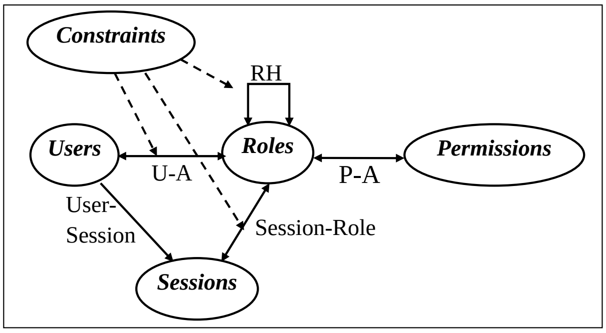


Figure 1. RBAC relational diagram.

3 Context-Aware Access Control Model

RBAC model has four main elements including users, roles, permissions and sessions. There are two sets of assignments, user assignment (U-A) and permission assignment (P-A). U-A is a relation that maps some roles to each user, and P-A assigns some permissions to each role. Each user can create some sessions and in each session, he/she has some active roles from their assigned roles.

$RBAC_0$ is the core of RBAC, in which users are associated with roles (U-A) and roles with permissions (P-A). $RBAC_1$ extends $RBAC_0$ by adding role hierarchy. In $RBAC_2$, constraints on role assignments and role activations are added to the core model. The complete RBAC model, i.e., $RBAC_3$, is the combination of the $RBAC_1$ and $RBAC_2$ models. Figure 1 shows the entity-relationship diagram of $RBAC_3$ (which we simply refer to as RBAC).

Although our model, iCAP, is role-based, it is not really an extension of RBAC. Its definition of roles, user-role assignment, and permission-role assignment are different. Roles in RBAC accord with the specific real roles in the organization. However, in iCAP, roles are not actually equal to real roles in the organization. In fact, roles in this model are sets of permissions. Hence, the role concept in iCAP is more abstract than that of RBAC.

iCAP is a context-aware model, because context information affects assigning roles to users and activating permissions of users' roles in each request.

3.1 Model Description

There are eight elements in the iCAP model as Users, Roles, Prms, Sessions, LTC, STC, RAC and RPC.

- ***Users*** is a set of current users. Every user has some roles in each session which are assigned according to context information.
- ***Roles*** is a set of predefined roles in the system. Role hierarchies in the form of arbitrary partial

order are defined among roles. The hierarchy implies inheritances of permissions and users' memberships as well. Each role has some permissions which are activated in specific conditions.

- ***Prms*** is a set of permissions. Each permission is a pair of object and access right such as $\langle book, read \rangle$.
- ***Sessions*** is a set of sessions. A user can create a session and obtain some accesses to objects during the session. Each session is assigned to exactly one user; however, each user can create more than one session (similar to RBAC).
- ***LTC-Set*** is a set of Long-Term Contexts (LTC). Long-term contexts do not change during a session with very high probability. Taking ts as the average session lifetime and μ as an integer variable, LTCs are the contexts whose values do not change during $\mu.ts$. So, μ should be chosen as high as possible, to ensure that the probability of context changes during a session is insignificant. Selection of LTCs depends on the system and session lifetime. Assume average session time is less than 1 or 2 hours and μ is 3, so we can select date, age, and finger-print as LTCs.
- ***STC-Set*** is a set of Short-Term Contexts (STC). A short-term context may change during a session frequently. Considering the previous example, time, location, and CPU load are Short-Term Contexts. As another example, if average session time is about 2 days and μ is 3, date would be a Short-Term Context.
- ***RAC (Role-Assignment Condition)*** is the mapping of role r onto a set of long-term context conditions. In other words, some long-term context conditions are defined for every role. Roles are assigned to the session's user according to their assignment conditions and current LTC information at the beginning of a session.
- ***RPC (Role-Permission Condition)*** is the mapping of role r and permission p to a set of short-term context conditions. It means roles' permission can be activated if short-term context conditions are satisfied. For every request of a user in a session, if at least one of its assigned roles has the requested permission and its conditions are satisfied, the access permission would be granted to the user.

According to the above descriptions, contextual constraints are applied to the model in two levels. Firstly, Long-Term Context constraints are applied to the role-hierarchy and session-role assignments. Secondly, the role-permission activations are limited with STCs.

Using iCAP, everyone who wants to use services in the environment, is known as a user with some roles according to LTC information, even he/she is not known before. Then, when the user asks for an access, the decision will be made considering the user's roles, requested permission conditions, and STC information. Therefore, iCAP succeeds in controlling every user's accesses dynamically when conditions are changed.

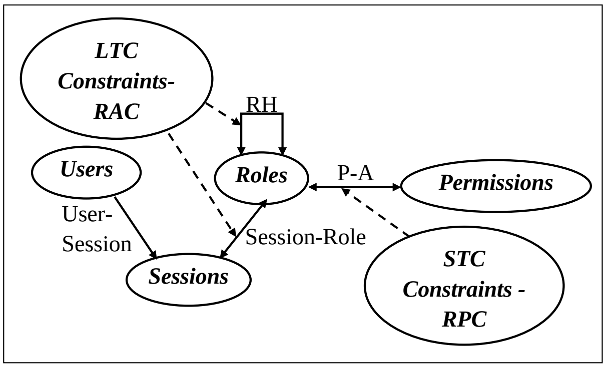


Figure 2. iCAP relational diagram.

The iCAP relational diagram is depicted in Figure 2. In the rest of this section, details of all parts of our model are described.

3.1.1 Context

In iCAP, context predicate is represented as the triple $\langle contexttype, contextrelater, value \rangle$ following the Gaia project's proposition [14]. In fact, context predicate determines the value of a context type according to the relater. A context predicate is used to specify both context information and context conditions. When it is used as a context information, its relater is "=" and it shows current value of the context type. When it is used as a context condition, its relater can be taken from the defined relaters in *CtxRelaterSet*, considering the context type. For example, $\langle Age, " > ", 10 \rangle$ means if user's age is greater than 10, the context condition is satisfied.

As it is already mentioned, for providing dynamic role assignments and permission activations, context set (*CtxSet*) is divided into Long-Term Context Set (*LTC-Set*) and Short-Term Context Set (*STC-Set*), which is shown in (1). When the value of a context type does not change in a session with a high probability, it is therefore defined as a long-term context type; otherwise, it is known as a short-term context type. Authentication context types (such as finger-print, ID card, password, certificate, etc) can be considered as different types of long-term contexts.

$$CtxSet = LTC\text{-}Set \cup STC\text{-}Set \quad (1)$$

$$LTC\text{-}Set \subseteq LTC\text{-}TypeSet \times CtxRelaterSet \times LTC\text{-}ValSet \quad (2)$$
$$LTC\text{-}TypeSet = Set\ of\ long\text{-}term\ context\ types$$
$$CtxRelaterSet = \{"=", "\neq", ">", "<", "\geq", "\leq"\}$$
$$LTC\text{-}ValSet = possible\ values\ of\ long\text{-}term\ context\ types$$

$$STC\text{-}Set \subseteq STC\text{-}TypeSet \times CtxRelaterSet \times STC\text{-}ValSet \quad (3)$$
$$STC\text{-}TypeSet = Set\ of\ short\text{-}term\ context\ types$$
$$CtxRelaterSet = \{"=", "\neq", ">", "<", "\geq", "\leq"\}$$
$$STC\text{-}ValSet = possible\ values\ of\ short\text{-}term\ context\ types$$

According to the relation (2), *LTC-Set* is a set of long-term context predicates, while every element of the set contains a long-term context type (an element of *LTC-TypeSet*), a context relater (an element of *CtxRelaterSet*) and its possible value (an element of *LTC-ValSet*). Likewise, the definition of *STC-Set* is shown in relation (3), which defines *STC-Set* as a subset of short-term context predicates. *STC-Set* is a subset of the Cartesian product of short-term context type set (*STC-TypeSet*), context relater set, and short-term context value set (*STC-ValSet*). For every context type, the set of possible values is a subset of allowable context values.

Every context type is related to only one type of the entities. For example, location and fingerprint are related to users, or temperature and time belong to the environment. In iCAP, the entity is a user or the environment that its context information potentially affects the authorization. Relation (4) shows the entity set (*EntSet*) as the union of the environmental entity set (*EnvEntity*) (which the environment (*env*), is the only element of it) and user set (*Users*).

$$EntSet = Users \cup EnvEntity \quad (4)$$
$$Users = Set\ of\ users$$
$$EnvEntity = \{env\}$$

Since every element of *LTC-Set* is related to either Users or *EnvEntity*, *LTC-Set* can be divided into two sets of environmental LTCs (*E-LTC-Set*) and user-related LTCs (*U-LTC-Set*). Similarly, *STC-Set* is the union of environmental STCs (*E-STC-Set*) and user-related STCs (*U-STC-Set*), as shown in relations (5) and (6), respectively.

$$LTC\text{-}Set = E\text{-}LTC\text{-}Set \cup U\text{-}LTC\text{-}Set \quad (5)$$
$$STC\text{-}Set = E\text{-}STC\text{-}Set \cup U\text{-}STC\text{-}Set \quad (6)$$

According to the division of *CtxSet* into *LTC-Set* and *STC-Set*, current context information is categorized into the following groups; *LTCI*, which is current Long-Term Context Information, and *STCI*, that includes current Short-Term Context Information. Relation (7) shows the definition of *LTCI*, which is an element of the power set of the Cartesian product of *EntSet* and *LTC-Set*. Likewise, *STCI* is an element of the power set of the Cartesian product of the *EntSet* and *STC-Set* (i.e., relation (8)). In the following relations, 2^S refers to the power set of the set *S*.

$$LTCI = 2^{(EntSet \times LTC\text{-}Set)} \quad (7)$$
$$STCI = 2^{(EntSet \times STC\text{-}Set)} \quad (8)$$

3.1.2 Role-Assignment Condition

At the beginning of a session, appropriate roles are assigned to the session's user according to the current *LTCI*. So the sufficient LTC conditions need to be defined for a role assignment. According to the relation (9), LTC condition set (*LTC-Cond*) is the Cartesian product of the power set of *U-LTC-Set* and the power set of *E-LTC-Set*. Also in (10), Role Assignment Condition (*RAC*) is a many-to-many mapping, LTC condition-to-role assignment relation. Hence, *rac* in (11) is the mapping of role r onto a set of LTC conditions (*LTC-Cond*). Formally, $rac(r) = \{ltcset \in 2^{LTC-Cond} | (r, ltcset) \in RAC\}$.

$$LTC\text{-}Cond = 2^{U\text{-}LTC\text{-}Set} \times 2^{E\text{-}LTC\text{-}Set} \quad (9)$$
$$RAC \subseteq Roles \times 2^{LTC\text{-}Cond} \quad (10)$$
$$rac(r \in Roles) \rightarrow 2^{2^{LTC\text{-}Cond}} \quad (11)$$

Henceforth, for every $(r, ltcset) \in RAC$, $rac(r).lcond$ refers to every $ltc \in ltcset$. Also for every $(u\text{-}set, e\text{-}set) \in ltc$, $rac(r).lcond(ltc).uset$ refers to *u-set*, and $rac(r).lcond(ltc).eset$ refers to *e-set*.

Logically, every $ltc \in LTC\text{-}Cond$ is the logical conjunction of its elements (i.e., a set of some user-related LTC conditions and a set of environmental LTC conditions). Moreover, every $cset \in 2^{LTC\text{-}Cond}$ is the logical disjunction of its elements. It means if at least one member of *cset* (which is a member of the *LTC-Cond* set) is satisfied, the role is assigned to the user. Logical description of *rac* for every role $r \in Roles$ is as follows:

$$rac(r) = \bigvee_{i \in N} \left[\bigwedge_{j \in N} u\text{-}ltc_{ij} \wedge \bigwedge_{k \in N} e\text{-}ltc_{ik} \right],$$
$$u\text{-}ltc_{ij} \in U\text{-}LTC\text{-}Set, e\text{-}ltc_{ik} \in E\text{-}LTC\text{-}Set$$

As an example, assume the role "teacher" is assigned to a session user in two different situations: 1) when his/her fingerprint is "f1" and ID is "122", and he/she tries on a weekday; or 2) when his/her fingerprint is "f2", and he/she tries on a weekday and not in summer.

rac (teacher) = {{<Fingerprint,"=", f1> and <ID,"=", 122345>}
***and** {<Day,"=", weekday>}}*
***OR** {{<Fingerprint, "=", f2>}*
***and** {<Day, "=", weekday> and <Season, "≠", summer>}}*

Figure 3. Role-Assignment-Condition Example

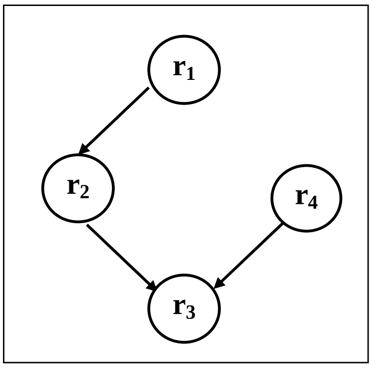


Figure 4. A sample of role hierarchy.

So, as shown in Figure 3, role assignment conditions for the role "teacher" is as follows:

3.1.3 Role Hierarchy

Role hierarchies in iCAP are modeled by a Directed Acyclic Graph (DAG), whose roles are its vertexes. Similar to RBAC, role hierarchy defines an inheritance relationship among roles. In the graph, each edge represents an inheritance relationship between two roles. As an example in Figure 4, there is an indirect relationship from r_1 to r_3 that means r_1 is the parent of r_3, or r_1 dominates r_3. It is obvious that, every role dominates itself.

Role hierarchies in RBAC support the concept of multiple inheritances, which provides the ability to inherit permission from two or more role sources and to inherit user membership from two or more role sources. Thus, according to RBAC [7], if r_1 inherits r_2 ($r_1 \succeq r_1$), all permissions of r_2 are also permissions of r_1, and all users of r_1 are users of r_2 as well.

Likewise, inheritance relationship in iCAP defines both the permission inheritance and user inheritance relationships. In other words, if r_1 inherits r_2:

- According to the user inheritance, all users of r_1 are users of r_2. This principle is applied during the role assignment process. If role r_1 is assigned to a user in a session, consequently role r_2 is assigned to the user in that session.
- Considering the permission inheritance, r_1 inherits all permissions of r_2. It means, if r_1 is authorized for permission in a specific situation, definitely, r_2 is authorized for the permission in that situation.

3.1.4 Session Assignment

Each session belongs to exactly one user. The relation (12) demonstrates *S-U* as the mapping of session s onto a user u.

$$S\text{-}U(s \in Sessions) \rightarrow Users \tag{12}$$

After a session starts, authorized roles are assigned to the session's user. *S-R* in (13) is the mapping of session s onto a set of the roles that are authorized. Figure 5 shows the definition of *S-R(s)*. Authorized roles contain *direct-roles* and *indirect-roles*. A role rl is a member of *direct-roles* when there is a $ltc \in rac(r_1).lcond$ (line 1 in the relation), that $LTCI$ satisfies its users-related context conditions (line 2 in the relation) and environmental assignment conditions (line 3 in the relation). According to the user inheritance relationship when a role is assigned to a user, its children in the role hierarchies are assigned to the user as well. Hence, *indirect-roles* are dominated by direct roles.

$$S\text{-}R(s \in Sessions) \rightarrow 2^{Roles} \tag{13}$$

Thus, *S-R(s)* includes roles which are assigned to the session's user dynamically when the session starts. These roles do not change during the session.

3.1.5 Role-Permission Condition

In iCAP, each role has some permissions, which are activated when their preconditions are satisfied. Permission activation conditions are short-term context conditions and they are defined statically as Role-Permission Conditions in the core of the model. In other words, iCAP activates roles' permissions according to the user-related and environmental STC conditions that have been defined before, and the current *STCI*. According to the relation (14)), STC condition set (*STC-Cond*) is defined as the Cartesian product of the power set of *U-STC-Set* and the power set of *E-STC-Set*. Therefore, in (15), Role Permission Condition (*RPC*) is the Cartesian product of roles, permissions, and the power set of STC conditions.

$$STC\text{-}Cond = 2^{U\text{-}STC\text{-}Set} \times 2^{E\text{-}STC\text{-}Set} \tag{14}$$

$$RPC \subseteq Roles \times Prm \times 2^{STC\text{-}Cond} \tag{15}$$

Condition definition must follow the permission inheritance relationship. Therefore, if role r_1 dominates role r_2:

- defined permission set of r_2 must be a subset of defined permission set of r_1, and
- for every defined permission of r_2, the permission activation condition of r_1 must not be more limited than the permission activation condition of

$S\text{-}R(s)= direct\text{-}roles(s) \cup indirect\text{-}roles(s)$

$direct\text{-}roles(s) =\{rl \in Roles|$

1.$\exists ltc \in rac(rl).lcond$

2.$[\forall t \in LTC\text{-}TypeSet, r \in CtxRelaterSet, v \in LTC\text{-}ValSet$

$[\langle t, r, v\rangle \in rac(rl).lcond(ltc).uset \wedge \langle S\text{-}U(s), t, r, v\rangle LTCI]$

$\wedge$

3.$\forall t' \in LTC\text{-}TypeSet, r' \in CtxRelaterSet, v' \in LTC\text{-}ValSet$

$[\langle t', r', v'\rangle \in rac(rl).lcond(ltc).eset \wedge \langle env, t', r', v'\rangle LTCI]]\}$

$indirect\text{-}roles(s) = \{rl' \in Roles | rl' \in direct\text{-}roles(s)[rl \succ rl']\}$

Figure 5. *S-R* Relation

r_2. Because, if role r_2 is authorized for a permission in a specific situation, role r_1 must be authorized for the permission in the same situation.

Considering the aforementioned definitions, permission activation conditions are divided into two parts for every role and permission:

- *Explicit conditions* are explicitly defined for activating permission p of role r in *RPC*.
- *Implicit conditions* are inherited from parents of the role r for the permission p. Note that, these conditions are the explicit conditions of the parent roles.

RPC is the collection of explicit conditions for activating each role's permission. Therefore, *activation-cond* is defined to return explicit and implicit conditions for activating a role's permission. According to (16), *activation-cond* is the mapping of role r and permission p onto a set of STC conditions, which is formally defined in (17).

$$activation\text{-}cond(r \in Roles, p \in Prm) \rightarrow 2^{2^{STC\text{-}Cond}} \quad (16)$$

$$activation\text{-}cond(r \in Roles, p \in Prm) = \{stcset \in 2^{STC\text{-}Cond} | \forall r' \succeq r, (r', p, stcset) \in RPC\} \quad (17)$$

Henceforth, for every $(r, p, stcset) \in RPC$, *activation-cond(r,p).xcond* refers to *stcset*, and for every $stc \in stcset$, *activation-cond(r,p).xcond(stcset).scond* refers to *stc*. For every $(u\text{-}set, e\text{-}set) \in stc$, *activation-cond(r,p).xcond(stcset).scond(stc).uset* refers to*u-set* , and *activation-cond(r,p).xcond(stcset).scond(stc).eset* refers to *e-set*.

Logically, similar to *LTC-Cond*, every $stc \in STC\text{-}Cond$ is the logical conjunction of its elements (i.e., a set of some user-related STC conditions and a set of environmental STC conditions). Also, every $cset \in 2^{STC\text{-}Cond}$ is the logical disjunction of its elements. Moreover, *activation-cond(r, p)* is the logical conjunction of explicit conditions for activating the permission p for every role $r' \succeq r$. It means permission p is activated for role r, if for every role $r' \succeq r$ and $stc \in 2^{STC\text{-}Cond}$ that $(r', p, stc) \in RPC$, at least one member of *stc* (which is a member of the *STC-Cond* set) is satisfied. Logical description of *activation-cond*, for every role $r \in Roles$ and permission $p \in Prm$, is as follows:

$$activation\text{-}cond(r,p) = \bigwedge_{r' \succeq r} \left[\bigvee_{i \in N} \left[\bigwedge_{j \in N} u\text{-}stc_{ij} \wedge \bigwedge_{k \in N} e\text{-}stc_{ik} \right] \right],$$

$$u\text{-}stc_{ij} \in U\text{-}STC\text{-}Set, e\text{-}stc_{ik} \in E\text{-}STC\text{-}Set$$

Table 1is an example of *RPC* definition for the defined roles in Figure 4. Each u_i refers to a user-related STC condition, and each e_i refers to an environmental STC condition.

3.1.6 Access Control

After a session starts and authorized roles are assigned to the session's user, for every access request of the user, iCAP checks conditions of this permission in *RPC* in comparison with *STCI*. Actually, iCAP checks the availability of the permission for only root roles in the assigned role hierarchies (i.e., the *direct-roles* set in Figure 5). Because, following the permission inheritance principle, if a role is authorized for a permission, its children are also authorized for the permission. Moreover, if at least one activated role is authorized for the permission, the requested access is permitted.

In (18), Access Decision Function (*ADF*) is described. *ADF* is the mapping of session s and permission p onto the decision set (which includes "*Grant*" and "*Deny*"). Showing in Figure 6, *ADF(s, p)* grants the permission p to the session's user if there is an assigned role *rl* (which is a root role in assigned role hierarchies) that its activation conditions are met in *STCI*. In other words, a permission p is activated for role *rl* if for every $stcset \in activation\text{-}cond(rl, p).xcond$ (line 1), there exists a $stc \in activation\text{-}cond(r, p).xcond(stcset).scond$ (line 2), that *STCI* satisfies its user-related context conditions (line 3) and environmental context conditions (line 4).

$$ADF(s \in Sessions, p \in Prm) \rightarrow \{Grant, Deny\} \quad (18)$$

Table 1. A sample of Role-Permission Conditions.

activation-cond(role,prm)	Implicit Conditions	Explicit Conditions
activation-cond(r_1,*p*)	{}	{{(u_1 *and* u_2) *and* (e_1 *and* e_2)} **or** {(u_1) *and* (e_2 *and* e_3)}}
activation-cond(r_2,*p*)	*activation-cond*(r_1,*p*)	{(u_3 *and* u_4) *and* (e_3 *and* e_4)}
activation-cond(r_4,*p*)	{}	{{(u_5) *and* (e_6)} **or** {(u_6) *and* (e_6 *and* e_7)}}
activation-cond(r_3,*p*)	*activation-cond*(r_1,*p*) *and* *activation-cond*(r_4,*p*)	{(u_5 *and* u_6) *and* (e_5 *and* e_6)}

$$
ADF(s,p) = \begin{cases}
\textbf{Grant}, \quad \text{if } \exists rl \in \textit{direct-roles}(s) \\
1. \quad [\forall stcset \in \textit{activation-cond}(rl,p).xcond \\
2. \quad\quad [\exists stc \in \textit{activation-cond}(rl,p).xcond(stcset).scond \\
3. [\forall \langle t \in STC\text{-}TypeSet, r \in CtxRelaterSet, v \in STC\text{-}ValSet \rangle \in \textit{activation-cond}(r,p).xcond(stcset).scond(stc).uset \\
\quad\quad\quad [\langle S\text{-}U(s), t, r, v \rangle \in STCI]] \\
\wedge \\
4. [\forall \langle t' \in STC\text{-}TypeSet, r' \in CtxRelaterSet, v' \in STC\text{-}ValSet \rangle \in \textit{activation-cond}(r,p).xcond(stcset).scond(stc).eset \\
\quad\quad\quad [\langle env, t', r', v' \rangle \in STCI]]]] \\
\textbf{Deny}, \quad \textit{otherwise}
\end{cases}
$$

Figure 6. *ADF* function.

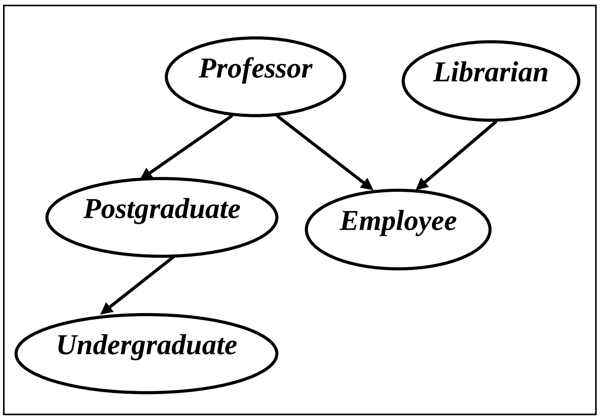


Figure 7. Role hierarchies of the library system.

Prms = { <ReferenceBooks, Reserving>, // as Res-Ref
<ReferenceBooks, Borrowing>, // as Brw-Ref
<ReferenceBooks, Extending>, // as Ext-Ref
<ReferenceBooks, TakingOut>, // as Tko-Ref
<ReferenceBooks, Adding>, // as Add-Ref
<ReferenceBooks, Deleting>, // as Del-Ref
<CommonBooks, Reserving>, // as Res-Com
<CommonBooks, Borrowing>, // as Brw-Com
<CommonBooks, Extending>, // as Ext-Com
<CommonBooks, TakingOut>, // as Tko-Com
<CommonBooks, Add>, // as Add-Com
<CommonBooks, Delete> // as Del-Com
}

Figure 8. Permission set of the library system.

U-LTC-TypeSet = {Fingerprint, IP-Address, CardID, Card-Pass}
E-LTC-TypeSet = {Season}
U-STC-TypeSet = {Location, BrwRefID (Browed Reference book ID), BrwRefNo (Number of Browed Reference book), BrwComID (Browed Common book ID), BrwComNo (Number of Browed Common book), Delay, ResRefID (Reserved Reference book ID), ResComID (Reserved Common book ID)}
E-STC-TypeSet = {Date, Time, Day}

Figure 9. Context sets of the library system.

4 Case Study

In this section, the access control of a university library is modeled using iCAP. Consider there are two types of objects in the library, namely reference books and common books. Users can reserve and borrow books and also extend their borrowed books or take them out of the library. Figure 7 demonstrates the role hierarchies in the system, which covers the *Undergraduate* student, *Postgraduate* student, *Professor*, *Employee*, and *Librarian* roles. Permission set is also shown in Figure 8.

Average session time is assumed around five hours. So, as Figure 9 shows, required context types are divided into four types of user-related and environmental long-term contexts, and user-related and environmental short-term contexts.

In Figure 10, required conditions for role assignments are defined according to long-term contexts.

```
activation-cond (Professor, Res-Ref) = {{}, {}}
activation-cond (Professor, Brw-Ref) = {{<BrwRefNo,"<", 3>},
    {<Day,"=", Weekday> and <Time, "<", EndTime> and
    <Time, ">", StartTime>}}
activation-cond (Professor, Ext-Ref) = {{<BrwRefID, "=", RefID>},
    {<Day,"=", Weekday> and <Time, "<", EndTime> and
    <Time, ">", StartTime>}}
activation-cond (Professor, Tko-Ref) = {{<BrwRefID, "=", RefID>},
    {<Day,"=", Weekday> and <Time, "<", EndTime> and
    <Time, ">", StartTime>}}
activation-cond (Professor, Res-Com) = {{}, {}}
activation-cond (Professor, Brw-Com) = {{<BrwComNo,"<", 5>},
    {<Day,"=", Weekday> and <Time, "<", EndTime> and
    <Time, ">", StartTime>}}
activation-cond (Professor, Ext-Com) = {{<BrwComID, "=", ComID>},
    {<Day,"=",Weekday> and <Time, "<", EndTime> and
    <Time, ">", StartTime>}}
activation-cond (Professor, Tko-Com)={{<BrwComID,"=", ComID>},
    {<Day,"=",Weekday> and <Time, "<", EndTime>
    and <Time, ">", StartTime>}}
```

Figure 11. Activation conditions for the *Professor* role.

```
activation-cond (Postgraduate, Res-Ref) =
    activation-cond (Professor,Res-Ref) and {{<Delay,"=', 0>}, {}}
activation-cond (Postgraduate, Brw-Ref) =
    activation-cond (Professor, Brw-Ref) and
    {{<BrwRefNo,"<", 1> and <ResRefID, "=", RefID>
    and <Delay, "=', 0>}, {}}
activation-cond (Postgraduate, Tko-Ref) =
    activation-cond (Professor, Tko-Ref) and {{}, {}}
activation-cond (Postgraduate, Res-Com) =
    activation-cond (Professor, Res-Com) and {{}, {}}
activation-cond (Postgraduate, Brw-Com) =
    activation-cond (Professor, Brw-Com) and {{<BrwComNo,"<",4>}, {}}
activation-cond (Postgraduate, Ext-Com) =
    activation-cond (Professor, Ext-Com) and {{}, {}}
activation-cond (Postgraduate, Tko-Com) =
    activation-cond (Professor, Tko-Com) and{{}, {}}
```

Figure 12. Activation conditions for the *Postgraduate* role.

For instance, the *Undergraduate* role can be assigned to a user, when his/her CardID and Card-Pass are equal to the defined values in *rac(Undergraduate)* and he/she does not try in summer.

Figures 11 to 15 show the activation conditions for the system roles' permissions. Assume professors can borrow at most three reference books and five common books; they can take them out of the library and extend the borrowing time on their books on loan (Figure 11).

Considering Figure 12, a postgraduate student can borrow at most one reference book, if he/she has already reserved it and does not have any overdue loans. In a similar situation, they can take the book out of the library. Postgraduate students can borrow at most four common books and extend the borrowed time. They can also take borrowed books out of the library.

Undergraduate students need to reserve common books and reference books, firstly. Then they are allowed to borrow them. An undergraduate student can borrow one reference book and three common books at the same time, if he/she is in the library,

```
activation-cond(Undergraduate, Res-Ref) =
    activation-cond (Postgraduate, Res-Ref)and {{}, {}}
activation-cond (Undergraduate, Brw-Ref)=
    activation-cond (Postgraduate, Brw-Ref) and
    {{<Location,"=",library>},{}}
activation-cond(Undergraduate, Res-Com) =
    activation-cond (Postgraduate, Res-Com) and
    {{<Delay, "=', 0>}, {}}
activation-cond (Undergraduate, Brw-Com) =
    activation-cond (Postgraduate, Brw-Com) and
    {{<BrwComNo,"<", 3>    and <ResComID, "=", ComID>
    and <Delay, "=', 0> and <Location, "=", library>}, {}}
activation-cond(Undergraduate, Ext-Com)=
    activation-cond (Postgraduate, Ext-Com) and
    {{<Delay, "=', 0>}, {<Date, "≤", DeliveryDate>}}
activation-cond(Undergraduate, Tko-Com)=
    activation-cond (Postgraduate, Tko-Com) and
    {{<Delay, "=', 0>}, {}}
```

Figure 13. Activation conditions for the *Undergraduate* role.

```
activation-cond (Librarian, Res-Ref) = {{<Delay, "=', 0>, {}}
activation-cond (Librarian, Brw-Ref) = {{<Location, "=", library> and
    <BrwRefNo,"<", 1> and <ResRefID, "=", RefID> and
    <Delay, "=', 0>}, {<Day,"=", Weekday>}}
activation-cond (Librarian, Tko-Ref) = {{<Delay, "=', 0> and
    <BrwRefID, "=", RefID>}, {<Day,"=", Weekday> and
    <Time, "<", EndTime> and <Time, ">", StartTime>}}
activation-cond (Librarian, Res-Com) = {{}, {}}
activation-cond (Librarian, Brw-Com) = {{<BrwComNo,"<", 3>  and
    <ResComID, "=", ComID> and <Location, "=", library>},
    {<Day,"=", Weekday>}}
activation-cond (Librarian, Ext-Com) = {{<BrwComID, "=", ComID>},
    {<Date,"≤",DeliveryDate>}}
activation-cond (Librarian, Tko-Com) = {{<BrwComID,"=", ComID>},
    {<Day,"=",Weekday>}}
activation-cond (Librarian, Add-Ref) = {{}, {<Day,"=", Weekday>  and
    <Time,"<",EndTime> and <Time, ">", StartTime>}}
activation-cond (Librarian, Del-Ref) = {{}, {<Day,"=", Weekday> and
    <Time, "<", EndTime> and <Time, ">", StartTime>}}
activation-cond (Librarian, Add-Com) = {{}, {<Day,"=", Weekday> and
    <Time,"<",EndTime> and <Time, ">", StartTime>}}
activation-cond (Librarian, Del-Com) = {{}, {<Day,"=", Weekday> and
    <Time, "<",EndTime> and <Time, ">", StartTime>}}
```

Figure 14. Activation conditions for the *Librarians* role.

and has returned all borrowed books by the due date. Undergraduates can only take common books out of the library. Figure 13 shows activation conditions for the *Undergraduate* role.

Figure 14 explains that librarians are able to add new books to the library list and delete out-of-order books from the list. Such actions can only take place when they are in the library and only during predefined working hours of weekdays. A librarian can borrow at most three common books and take them out on weekdays, if he/she has no overdue loans. Moreover, they are able to borrow one reference book after reservation, only when they are in the library and only on weekdays. However, they are not allowed to take the reference books out of the library. A librarian can extend borrowing time of common books before their due dates.

Finally, considering Figure 15, an employee can re-

rac (Teacher) = {{<Fingerprint,"=", f_1>}, {}} or {{<Fingerprint,"=", f_2>}, {}} or {{<Fingerprint,"=", f_3>}, {}}

rac (Postgraduate) = {{<CardID,"=", 84026> and <Card-Pass,"=", jsd4>}, {}} or
{{<CardID,"=", 84027> and <Card-Pass,"=", j4nt>}, {}} or
{{<CardID,"=", 84028> and <Card-Pass,"=", 8rh4>}, {}}

rac (Undergraduate) = {{<CardID,"=", 84110> and <Card-Pass,"=", frt5>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84111> and <Card-Pass,"=", j45u>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84112> and <Card-Pass,"=", 8rh4>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84113> and <Card-Pass,"=", 9r3r>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84114> and <Card-Pass,"=", mc46>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84115> and <Card-Pass,"=", fhj5>}, {<Season, "≠", Summer>}} or
{{<CardID,"=", 84116> and <Card-Pass,"=", jd6f>}, {<Season, "≠", Summer>}}

rac (Librarian) = {{<IP-Address,"=", 192.162.16.1> and <Fingerprint,"=", f4>}, {}} or
{{<IP-Address,"=", 192.162.16.2> and <Fingerprint,"=", f5>}, {}} or
{{<IP-Address,"=", 192.162.16.3> and <Fingerprint,"=", f6>}, {}} or
{{<IP-Address,"=", 192.162.16.4> and <Fingerprint,"=", f7>}, {}} or
{{<IP-Address,"=", 192.162.16.5> and <Fingerprint,"=", f8>}, {}}

rac (Employee) ={{<CardID,"=", 12120> and <Card-Pass,"=", frt5> and <IP-Address,"=", 192.162.34.1>}, {}} or
{{<CardID,"=", 12121> and <Card-Pass,"=", j45u> and <IP-Address,"=", 192.162.34.2>}, {}} or
{{<CardID,"=", 12122> and <Card-Pass,"=", 8rh4> and<IP-Address,"=", 192.162.34.3>}, {}} or
{{<CardID,"=", 12123> and <Card-Pass,"=", 9r3r> and <IP-Address,"=", 192.162.34.4>}, {}} or
{{<CardID,"=", 12124> and <Card-Pass,"=", mc46> and <IP-Address,"=", 192.162.34.5>}, {}}

Figure 10. Role-Assignment-Conditions for the library system.

serve reference books and common books on weekdays and during working hours, under the circumstances that he/she is in the library and does not have any overdue loans. Employees can borrow at most one reference book as the same constraints as the librarians. But they are not allowed to take the reference books out of the library. They can borrow two common books, after satisfying the permission activation conditions for both the *Librarian* and the *Professor* roles. Additionally, their borrowed books must not be overdue. Likewise, borrowed common books can be taken out of the library, when the permission activation conditions for the *Professor* and the *Librarian* roles are satisfied. An employee is not allowed to extend the borrowing time of the books.

Implicit conditions are inherited from the parents of each role in the role hierarchy. For example, the *Employee* role inherits explicit conditions of the *Professor* and the *Librarian* roles for all of its permissions. Hence, activating condition of the *Brw-Com* permission (borrowing a common book) for the *Employee* role is as follows:

activation-cond(Employee, Brw-Com) =
$activation\text{-}cond(Professor, Brw\text{-}Com)$**AND**
$activation\text{-}cond(Librarian, Brw\text{-}Com)$**AND**
$\{\{\langle BrwComNo, "<", 2\rangle and \langle Delay, "=", 0\rangle\}, \{\}\}$
activation-cond(Employee, Brw-Com) =
$\{\{\langle BrwComNo, "<", 5\rangle\}, \{\langle Day, "=", Weekday\rangle and$
$\langle Time, "<", EndTime\rangle and \langle Time, ">", StartTime\rangle\}\}$
AND
$\{\{\langle BrwComNo, "<", 3\rangle and \langle ResComID, "=", ComID\rangle$
$and \langle Location, "=", library\rangle\}, \{\langle Day, "=", Weekday\rangle\}\}$
AND $\{\{\langle BrwComNo, "<", 2\rangle and \langle Delay, "=", 0\rangle\}, \{\}\}$

It is clear that, some conditions may be repeated in the implementation (such as *BrwComNo* in the last

activation-cond (Employee, Res-Ref) =
activation-cond (Professor, Res-Ref) ***and***
activation-cond(Librarian, Res-Ref) ***and***
{{<Location, "=", library>}}, {<Day,"=", Weekday> and
<Time, "<", EndTime> and <Time, ">", StartTime>}}
activation-cond (Employee, Brw-Ref) =
activation-cond (Professor, Brw-Ref) ***and***
activation-cond (Librarian, Brw-Ref) ***and***
{{}, {}}
activation-cond (Employee, Res-Com) =
activation-cond (Professor, Res-Com) ***and***
activation-cond (Librarian, Res-Com) ***and***
{{<Delay, "=', 0> and <Location, "=", library>},
{<Day,"=", Weekday> and <Time,"<", EndTime>
and <Time, ">", StartTime>}}
activation-cond (Employee, Brw-Com) =
activation-cond (Professor, Brw-Com) ***and***
activation-cond (Librarian, Brw-Com) ***and***
{{<BrwComNo,"<", 2> and <Delay, "=', 0 >}, {}}
activation-cond (Employee, Tko-Com) =
activation-cond (Professor, Tko-Com) ***and***
activation-cond (Librarian, Tko-Com) ***and***
{{<Delay, "=', 0>}, {}}

Figure 15. Activation conditions for the *Employee* role.

example), but they are gathered just one time and so this problem does not increase the execution time.

Using these definitions, now we follow a scenario. Assume that Bob is a user. He connects to the library system, and starts a new session. First of all, he is assigned to a session, called s_1; i.e., $S\text{-}U(s_1) = Bob$. Suppose long-term context information at this time is as follows:

$$LTCI = \{\langle Bob, IP\text{-}Address, "=", 192.162.16.1\rangle, \langle Bob, Finger\text{-}Print, "=", f4\rangle, \langle Bob, CardID, "=", 12345\rangle, \langle Bob, Card\text{-}Pass, "=", jsd4\rangle\}$$

Regarding *LTCI* and *RAC*, assignment conditions of the *Postgraduate* and *Librarian* roles are satisfied. However, the *Undergraduate* role is assigned implicitly to this session, because it is the child of the *Postgraduate* role in the role hierarchies. Hence, session assigned roles are as follows: $S\text{-}R(s_1) = \{Postgraduate, Undergraduate, Librarian\}$. Assume *STCI* is as follows:

$$STCI = \{\langle Bob, BrwRefNo, "=", 0\rangle, \langle env, Day, "=", Friday\rangle, \langle Bob, Delay, "=", 0\rangle, \langle Bob, Location, "=", home\rangle, \langle Bob, ResRefID, "=", RefID\rangle\}$$

Now Bob sends a request for borrowing a reference book. The decision function (ADF) checks activation conditions of this permission for the *Librarian* and the *Postgraduate* roles. Role-Permission-Conditions for these roles and borrowing reference book permission are as follows:

activation-cond(Librarian, Brw-Ref) =
$\{\{\langle Location, "=", library\rangle and \langle BrwRefNo, "<", 1\rangle and \langle ResRefID, "=", RefID\rangle and \langle Delay, "=", 0\rangle\}, \{\langle Day, "=", Weekday\rangle\}\}$

activation-cond(Postgraduate, Brw-Ref) =
$activation-cond(Professor, Brw\text{-}Ref)$ **AND**
$\{\{\langle BrwRefNo, "<", 1\rangle and \langle ResRefID, "=", RefID\rangle and \langle Delay, "=", 0\rangle\}, \{\}\}$

activation-cond (Postgraduate, Brw-Ref) =
$\{\{\langle BrwRefNo, "<", 3\rangle\}, \{\langle Day, "=", Weekday\rangle and \langle Time, "<", EndTime\rangle and \langle Time, ">", StartTime\rangle\}\}$
AND
$\{\{\langle BrwRefNo, "<", 1\rangle and \langle ResRefID, "=", RefID\rangle and \langle Delay, "=", 0\rangle\}, \{\}\}$

Activation conditions of the *Brw-Ref* permission are not satisfied for the librarian role, but activation conditions of the permission for the *Postgraduate* role are satisfied. Thus, *ADF* assigns *Grant* to this request in the session and Bob can borrow the reference book.

As another example, consider *STCI* defined as follows:

$$STCI = \{\langle Bob, BrwRefNo, "=", 0\rangle, \langle Bob, Delay, "=", 0\rangle, \langle Bob, ResRefID, "=", RefID\rangle, \langle Bob, Location, "=", home\rangle, \langle env, Day, "=", Saturday\rangle\}$$

Now if Bob sends the request again, *ADF* assigns *Deny* to the request of borrowing a reference book. Since Bob's current location is home and it is Saturday, activation conditions of the *Brw-Ref* permission for the *Librarian* and the *Postgraduate* roles are not satisfied.

5 Architecture

Due to heterogonous characteristics of pervasive environments, authorization should be decentralized in such areas. Hence, PCEs are divided into different domains using different access control systems. Context gathering is an important part of the access control process in pervasive computing environments and can significantly affect the decision. In iCAP, for every session, roles are assigned to the session user according to the long-term contexts. Then, every request of the user is checked considering the session roles and short-term contexts. Therefore, access control can be enforced by two agents; Domain Authority (DA) and Session Agent (SA). DA creates a session for the user and assigns authorized roles to the session, while SA controls the user's accesses in the session. There is one DA in every domain which assigns sessions and roles to users. DA sets up an SA for each session. So, each session has its own SA that controls access requests. Figure 16 demonstrates components of the architecture.

Domain Authority (DA) generates sessions for users and provides long-term context information (*LTCI*). It assigns authorized roles to a session user at the beginning of the session based on the provided *LTCI* and prerequisite conditions of the role assignments. Hence, DA updates the *S-U* and *S-R* sets in the domain.

DA contains dynamic and static datasets. Static datasets include Role-Assignment Conditions (*RAC*), Role Hierarchies (*RH*), and Role-Permission Condition (*RPC*); also dynamic dataset is only composed of Session-Roles (*S-R*). The basic components of a DA are as follows:

- **Long-Term Context Manager** provides long-term context values from different sources, such as environmental and user-related sensors. Also interprets contexts values and stores them in the specific format as *LTCI*.
- **Session Manager** receives requests from session's users, assigns a session and an SA to every user. Then asks Dynamic User-Role Assigner to determine the activated roles of the session's user. It fills the *RPC* dataset of the SA according to the assigned roles.
- **Dynamic User-Role Assigner** assigns roles to the session with respect to *LTCI*, *RAC*, and RH, and then, fills S-R dataset.

Session Agent (SA) controls user's accesses in the session. It collects short-term context information

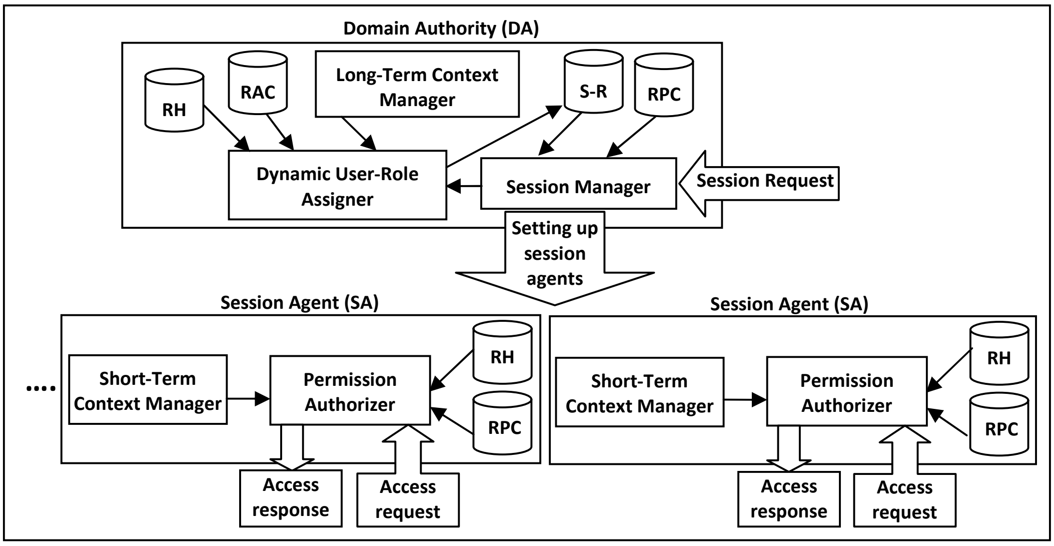


Figure 16. iCAP architecture.

and evaluates each user's request considering *RPC*, *STCI*, and assigned roles. If the requested permission is accepted by the decision function (*ADF*), the access is granted, otherwise the request is denied.

SA contains some dynamic datasets which are filled by DA, including Role-Permission Conditions (*RPC*) and Role Hierarchies (*RH*). These datasets are filled when an SA is set up. The main components of an SA are as follows:

- **Short-Term Context Manager** works as the same as Long-Term Context Manager in DA, but it collects Short-Term Contexts Information (*STCI*).
- **Permission Authorizer** makes appropriate decisions about the user's access requests according to session roles, role hierarchies, and the prerequisite context conditions for activating the requested permission.

5.1 Compatibility with XACML Standard

XACML [18] is the standard of OASIS for access control in context-aware environments. XACML consists of two models: policy language model and data-flow model.

Policy language model describes access control policies compatible with XML. The main components of the model are rule, policy, and policy set.

Figure 17 shows XACML data-flow diagram. Components of the model and their relationships are described in the diagram. The most important components of the model are as follows:

- PAP (Policy Administration Point) collects policies and makes them available to be used in PDP.
- PDP (Policy Decision Point) makes a decision for each request by gathering policies from PAP and necessary contexts from Context Handler.
- PEP (Policy Enforcement Point) manages access control.
- Context Handler gathers context information from PIP and resources, and changes XACML format of showing contexts to native format and vice versa.
- PIP (Policy Information Point) obtains requested contexts and sends them for context handler.

First of all, PEP receives an access request and sends it to Context Handler. Next, Context Handler notifies PDP with this request, and then, receives the needed context types from PDP and gathers the context values from PIP and resources, and then, transmits them to PDP. After that, PDP makes a response by fetching policies from PAP and sends the response to Context Handler. Finally, Context Handler transmits the request response to PEP and it obligates the user by this response.

The proposed architecture is compatible with XACML standard. There are equivalent components in XACML data-flow model with the proposed architecture.

Equivalent components of DA and data-flow model are PEP with Session Manager, Context Handler and PIP with Long-Term Context Manager, PDP with Dynamic User-Role Assigner and PAP with the datasets.

Figure 18a shows DA architecture corresponding to XACML. As can be seen in the figure, PEP (Session Manager) sends a request to Context Manager, and then Context Manager requests the list of required contexts from PDP (Dynamic User-Role Assigner).

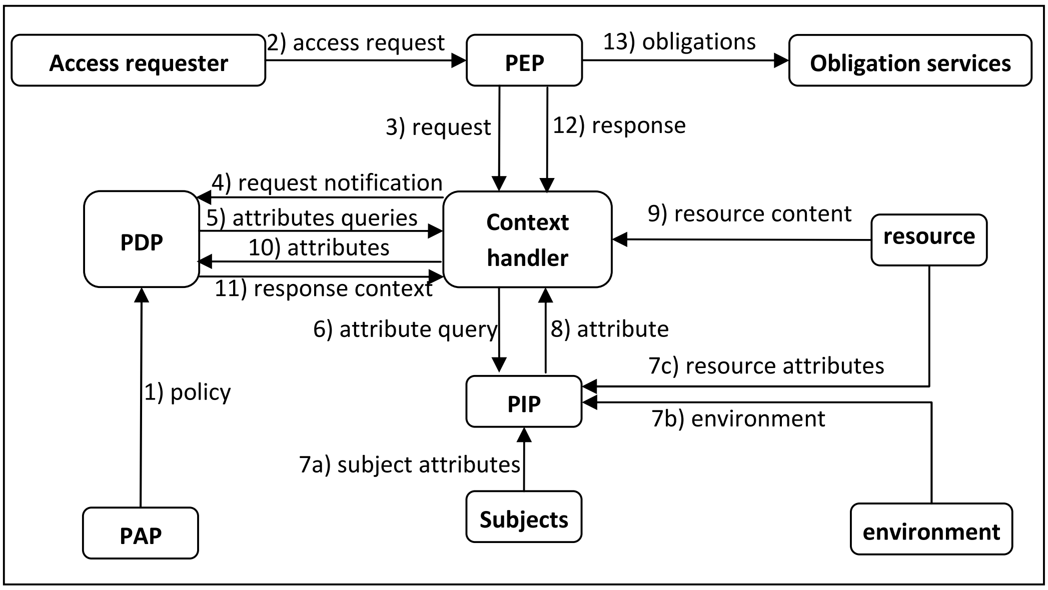


Figure 17. XACML data-flow diagram [18].

Since in iCAP model, PDP knows nothing about the needed context types at first, context handler has to collect all context type values and transmit them to PDP (because every role assignment condition should be checked). It causes a high overhead in the system, because all context type values are not usually needed for assigning authorized roles. To improve execution time, we change the sequence of PDP and Context Handler in the data-flow.

Figure 18b shows the improved DA data-flow considering the XACML data-flow. PEP sends a request to PDP and PDP requests the needed contexts from Context Handler. Context Handler receives context values from PIP and sends it to PDP. This process (steps 4 to 7) may repeat several times for gathering all required context values. But, whenever a context value rejects a role, there is no need to gather other context values in the role-assignment conditions.

Compatibility of SA architecture with XACML data-flow can be shown analogous to the DA architecture. As illustrated in Figure 19a, the SA components can be corresponded to the XACML components in the following way: PEP and PDP with Permission Authorizer, Context Handler and PIP with Context Manager, and PAP with the datasets. According to the data-flow, PEP notifies Context Handler for evaluating requests. Context Handler asks required short-term context values from PDP. Context Handler has to collect all context values that are defined as role-permission conditions. To improve execution time, similar to the DA data-flow, we change the SA data-flow to Figure 19b.

6 Evaluation

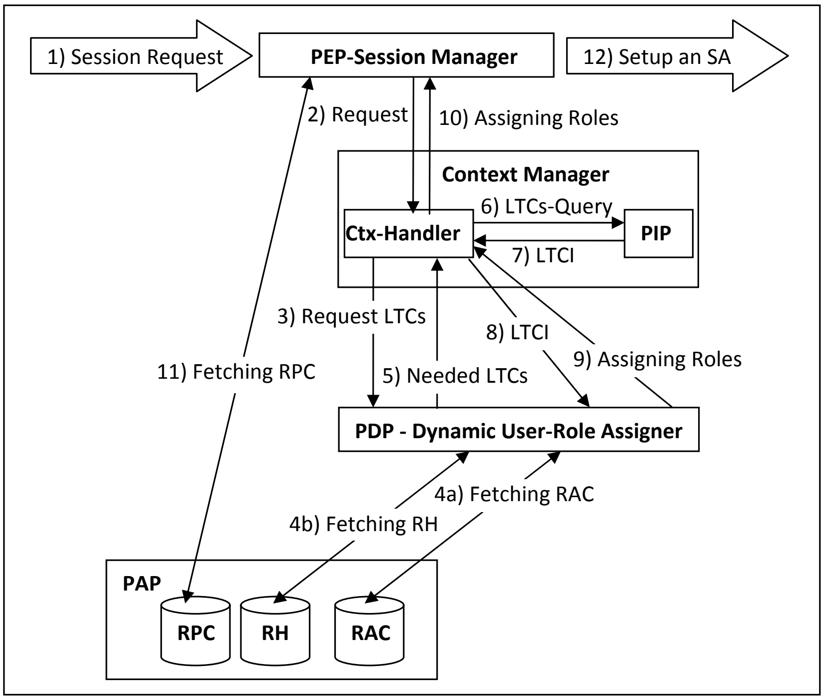
To evaluate the proposed model, we have collected a number of criteria by reviewing the evaluation procedure of various related work. We do not claim that the criteria set presented here is complete; however, due to their use in the evaluation of different models, such criteria can help us to compare iCAP with the relevant models.

6.1 Decidability and Complexity

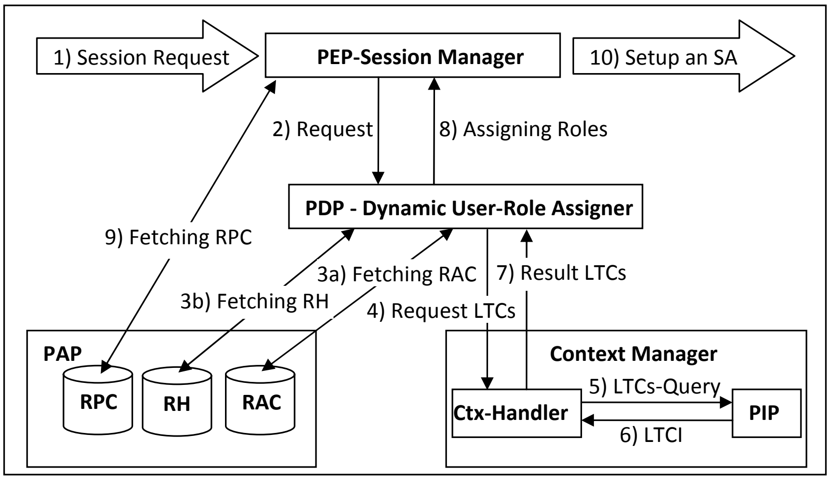
Decisions are made by *S-R* and *ADF* functions in iCAP. Thus, for proving the decidability of iCAP, we should prove these functions are decidable.

Role assignments and permission activations are based on contextual conditions. Due to the fact that the set of context types, context values and relaters are finite, it can be concluded that the role assignment conditions and role-permission activation conditions can be checked in a finite time. Likewise, these functions have been implemented and are executed in a finite time. Therefore, both *ADF* and *S-R* functions are decidable, and as a result iCAP is also decidable.

Figure 20 shows the algorithm of $S\text{-}R(s)$, which assigns authorized roles to the session s. The function gathers required user-related LTC values and environmental LTC values according to *RAC*. To optimize execution time of the algorithm, a sorted list of long-term contexts (SLTC) is generated. SLTC is an array of long-term context types which are arranged in order of the number of their occurrences in the conditions. For example, if the Date context type exists in more



a) DA data-flow based on XACML



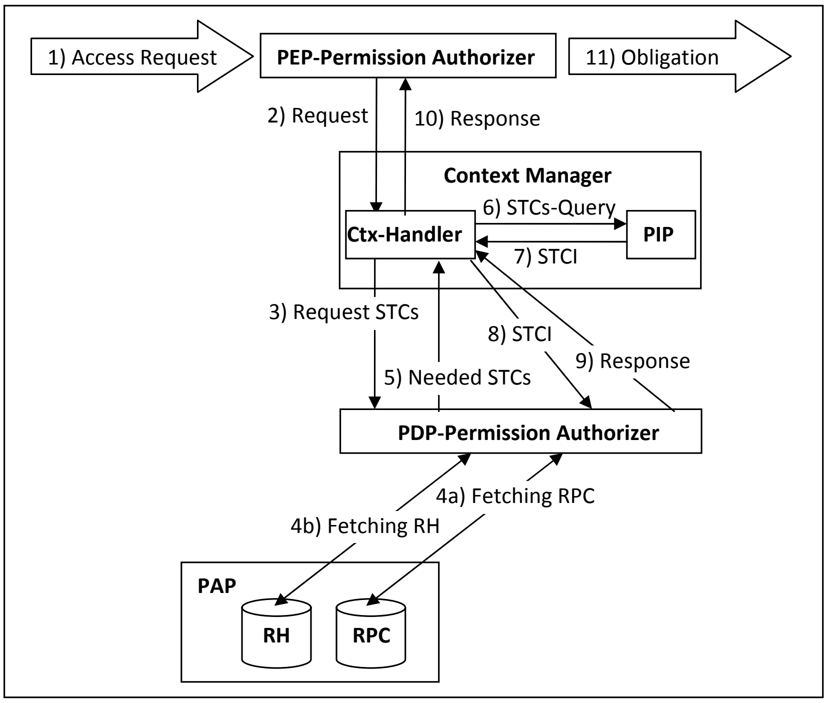
b) Improved DA data-flow

Figure 18. Domain authority (DA) data-flow.

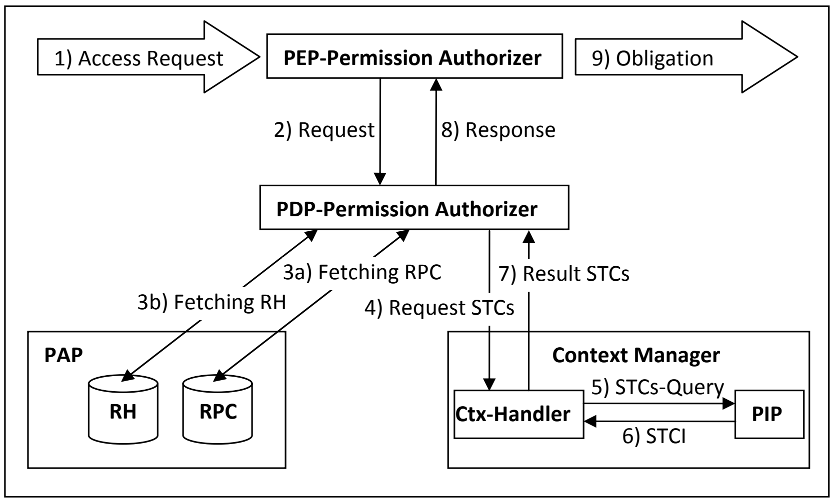
conditions than the Age context type, index of Date is less than index of Age in the array. Role-assignment conditions are checked, in order of the sorted array. Contexts which are checked sooner exist in more role assignment conditions. Thus, they affect acceptance or rejection of more roles. As a result, using SLTC, execution time is optimized.

Complexity of the *S-R* function depends on the number of long-term context types and roles. Consider l as the number of long-term context types (including user-related and environmental LTC types) and r as the number of roles. So, complexity of the first part of the algorithm which provides *direct-roles* is $O(r.l^2)$. The second part of the algorithm which provides *indirect-roles* uses depth-first manner for traversing the role hierarchy graph. If adjacency matrix is used for storing the graph, the complexity of depth-first traversing is $O(r^2)$. Thus, the complexity of the second part is $O(r^3)$. Hence, the complexity of *S-R* is $O(r.l^2 + r^3)$.

Figure 21 shows the algorithm *ADF* function which uses *activation-cond* function for finding activation conditions of permission p in session s. It checks activation conditions of p for every session role, which is a root role in the assigned role hierarchies. If the permission activation conditions are satisfied for at least one session role, *ADF* returns *Grant*, otherwise



a) SA data-flow based on XACML



b) Improved SA data-flow

Figure 19. Session Agent (SA) data-flow.

returns *Deny*. Complexity of *ADF* depends on the number of short-term context types and roles. Suppose n is the number of short-term context types (including user-related and environmental STC types) and r is the number of roles. Thus, the complexity of the algorithm is $O(r.n^3)$. Likewise, depth-first manner is used for traversing the role hierarchy graph in the *activation-cond* function. Thus, the complexity of the *activation-cond* function is $O(r^2)$. The complexity of *ADF* is $O(r.n^3 + r^2)$.

It is worthwhile to note that, in the context-aware models, the most time-consuming part is context evaluation. In iCAP, this part is performed in an optimal approach by dividing contexts into long-term and short-term contexts. LTCs are checked at the beginning of a session and thus they do not require to be checked during a session. However, STCs are evaluated in a session by receiving each access request. This is one of the main contributions of our model which makes it more applicable in comparison with other context-aware models.

```
Algorithm   S-R(Session s)
for every rl ∈ Roles do
  ltc-flag = true
  for every ltc ∈ rac(rl).lcond do
    for every e-ltc ∈ rac(rl).lcond(ltc).eset in order to SLTC do
      value = Fetch(e-ltc− > CtxType, LTCI)
      if value does not satisfy e-ltc then
        ltc-flag = false
        break
      end if
    end for
    for every u-ltc ∈ rac(rl).lcond(ltc).uset in order to SLTC do
      value = Fetch(u-ltc− > CtxType, LTCI)
      if value does not satisfy u-ltc then
        ltc-flag = false
        break
      end if
    end for
    if ltc-flag = true then
      Add rl to direct-roles
      break
    end if
  end for
end for
for every role rl′ ∈ Roles do
  if rl ≻ rl′ then
    Add rl′ to indirect-roles
  end if
end for
return the union of direct-roles and indirect-roles
```

Figure 20. Session role assignment algorithm.

6.2 Expressiveness

iCAP supports an unrestricted combination of context conditions for role assignment and role-permission activation. As described earlier, *RAC* stores the logical disjunction of the logical conjunction of user-related and environmental LTC conditions, for every role. Likewise, the logical disjunction of the logical conjunction of user-related and environmental STC conditions is stored in *RPC* for every role and its assigned permission.

iCAP can express the specifiable constraints in RBAC, including static and dynamic separation of duties. Furthermore, it supports the principle of least privilege. In RBAC, Static Separation of Duties (SSD) is applied on user-role assignment and role hierarchies [7]. For user-role assignment, a collection of pairs *(rs, n)* is defined, in which *rs* is a role set and *n* is a natural number greater than 1. For every *(rs, n)*, no user is authorized for *n* or more roles in *rs*. SSD on role hierarchies means two roles are mutually exclusive only if none of them inherits the other, and no role can inherit from both. Dynamic Separation of Duties (DSD) is also a collection of pairs *(rs, n)*. But it means that for each *(rs, n)*, no user can activate *n* or more roles in *rs* in the same session.

SSD on role hierarchies is applied in iCAP similar to RBAC. In iCAP roles are assigned to users dynamically at the beginning of the sessions. Hence, we can apply both SSD and DSD at the role assignment time. By defining a context type for each pair *(rs, n)*, we can apply SSD and DSD. For example, we have a pair of *(teacher, student, 2)*, it means that both teacher and student roles must not be assigned to a session user, simultaneously. A long-term context type is defined, namely *T-S*, which shows the number of assigned roles in the set of *teacher, student*. Then a user-related LTC condition is added to the teacher and student assignment conditions as $\langle T\text{-}S, " < ", 1\rangle$. When DA assigns roles to a user, it checks the value of *T-S*, which is the number of assigned roles in the set *teacher, student*; if it is less than 1 it assigns the role to the user.

According to the RBAC model, DSD properties provide extended support for the principle of least privilege. The principle supports the idea that every user may need different levels of permissions at different times, depending on the role being performed. These properties ensure that permissions do not persist beyond the time that they are required for performance of duty. This aspect of least privilege is often referred to as timely revocation of trust. Since iCAP supports

```
Algorithm   ADF(Permission p, Session s)
for every rl ∈ direct-roles(s) do
  stc-flag = true
  for every stcset ∈ activation-cond(rl, p).xcond do
    for every stc ∈ activation-cond(rl, p).xcond(stcset).scond do
      for every e-stc ∈ activation-cond(rl, p).xcond(stcset).scond(stc).eset
      in order to SSTC do
        value = Fetch(e-stc− > CtxType, STCI)
        if value does not satisfy e-stc then
          stc-flag = false
          break
        end if
      end for
      for every u-stc ∈ activation-cond(rl, p).xcond(stcset).scond(stc).uset
      in order to SSTC do
        value = Fetch(u-stc− > CtxType, STCI)
        if value does not satisfy u-stc then
          stc-flag = false
          break
        end if
      end for
      if stc-flag = true then
        break
      end if
    end for
    if stc-flag = false then
      break
    end if
  end for
  if stc-flag = true then
    return "Grant"
  end if
end for
return "Deny"


activation-cond(Role r, Permission p)
for every role r′ ⪰ r do
  if exists a stcset ∈ 2^(STC-Cond) AND(r′, p, stcset) ∈ RPC then
    Add stcset to rpc-set
  end if
end for
return rpc-set
```

Figure 21. Decision function algorithm.

DSD, similar to RBAC, it provides the extended support for the principle of least privilege.

6.3 Scalability

A suitable access control model for pervasive computing environment must necessarily take scalability issues into account [8]. In the iCAP model, the environment is divided into some domains which are administrated individually. In each domain, permissions are defined according to the domain requirements and can be activated according to the current context information. The set of roles are not fixed and new roles might be added to the model. Furthermore, users are unknown in the model, and policy rules which are applied to a user are based on context information. Hence, the model is intrinsically distributed that imposes scalability of the model.

6.4 Dynamicity

RBAC is a static model; it defines user-permission and role-permission assignments statically. Some extensions of RBAC such as DRBAC [11] and CGR-BAC [15]] tried to create a dynamic model based on RBAC by making it context-aware. However, most of them use context as conditions for role-permission assignment and they do not consider dynamic user-role assignment. iCAP not only controls accesses to the objects and activates the roles permissions according to the context information, but also assigns roles to users dynamically based on their context at the beginning of the sessions. In short, iCAP is dynamic in both user-role assignment and role-permission activation.

7 Conclusion

In this paper, iCAP, a context-aware access control model for PCEs was proposed. Since iCAP model is dynamic and scalable, it can control access of heterogeneous and unknown users in different situations. Context types are divided into two types of long-term and short-term ones, according to the average of changing periods of their values. Both long-term and short-term contexts are used in decision making. The model is role-based and dynamically assigns roles to users according to long-term contexts. Users' accesses are limited by short-term context information. The model was described in a formal manner, and also a real case study was presented to demonstrate the applicability of the model. Likewise, a complying architecture was proposed for the model, and the model was implemented based on the architecture. Finally, the model was evaluated based on some common criteria. Thus, expressiveness and complexity of the model was examined, also it is concluded that iCAP is applicable, decidable and dynamic.

References

[1] L. Kagal, T. Finin, and A. Joshi. Trust-based Security in Pervasive Computing Environments. *IEEE Computer*, 34:154–157, 2001.

[2] D. Saha and A. Mukherjee. Pervasive Computing: A Paradigm for the 21st Century. *IEEE Computer*, 36(3):25–31, 2003.

[3] J. L. Vivas, C. Fernandez-Gago, J. Lopez, and A. Benjumea. A Security Framework for a Workflow-based Grid Development Platform. *Computer Standards and Interfaces (being published by ELSEVIER)*, doi:10.1016/j.csi.2009.04.001, 2009.

[4] S. Singh and S. Bawa. A Privacy, Trust and Policy based Authorization Framework for Services in Distributed Environments. *The International Journal of Computer Science*, 2(2):85–92, 2007.

[5] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[6] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma. Context Sensitive Access Control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pages 111–119, Stockholm, Sweden, 2005.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, and R. Chandramouli. Proposed NIST Standard for Role Based Access Control. *ACM Transactions on Information and System Security*, 4:224–274, 2001.

[8] A. Kern and C. Walhorn. Rule Support for Role-Based Access Control. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pages 130–138, Stockholm, Sweden, 2005.

[9] W. Jih, S. Cheng, J. Y. Hsu, and T. Tsai. Context-aware Access Control on Pervasive Healthcare. In *Proceedings of the IEEE Workshop on Mobility, Agents, and Mobile Services (MAM)*, 2005 IEEE International Conference on e-Technology, e-Commerce, and e-Service, pages 21–28, Hong Kong, 2005.

[10] J. Al-Muhtadi, A. Ranganathan, R. H. Campbell, and M. D. Mickunas. Cerberus: A Context-Aware Security Scheme for Smart Spaces. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 489–496, Fort Worth, Texas, USA, 2003.

[11] 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, USA, 2004.

[12] U. Hengartner and P. Steenkiste. Access Control to Information in Pervasive Computing Environments. In *Proceedings of the 9th ACM Workshop on Hot Topics in Operating Systems (HotOSIX)*, volume 9, pages 157–162, Lihue, Hawaii, 2003. USENIX Association.

[13] F. Pu, D. Sun, Q. Cao, H. Cai, and F. Yang. Pervasive Computing Context Access Control Based on UCONABC Model. In *Proceedigns of the IEEE International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'06)*, pages 689–692, 2006.

[14] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. A Middleware Infrastructure for Active Spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[15] H. Shen and F. Hong. A Context-Aware Role-Based Access Control Model for Web Services. In *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2005)*, pages 220–223, 2005.

[16] J. H. Jafarian and M. Amini. CAMAC:AContext-AwareMandatory Access Control Model. *ISeCure: The ISC International Journal of Information Security*, 1(1):35–54, 2009.

[17] S. S. Emami, M. Amini, and S. Zokaei. A Context-Aware Access Control Model for Pervasive Computing Environments. In *Proceedings of the International Conference on Intelligent Pervasive Computing (IPC 2007)*, pages 51–56, Jijo Island, Korea, 2007. IEEE Computer Society.

[18] T. Moses. eXtensible Access Control Markup Language (XACML), Version 2.0, 2005. OASIS Standard, Technical Report, Available at http://docs.oasis-open.org Accessed 01, Mar 2009.

Sareh Sadat Emami received the BS degree in software engineering from Bu-Ali Sina University, Hamedan, Iran, in 2004 and the MS degree in IT- secure communication- from K.N. Toosi Uneversity of Technology, Tehran, Iran, in 2008. She is currently a member of the CERT group in Sharif Network Security Center (security research lab of Sharif University of Technology), Tehran, Iran. Her research interests are Computer Security, Ubiquitous/Pervasive Computing, Context-Awareness, and Cryptography.

Saadan Zokaei received the MS degree in electrical engineering from University of Tehran, Tehran, Iran and the PhD degree in electrical engineering from the Department of Communication and Information Technology, University of Tokyo, Japan in 1994. He is with the Department of Electrical Engineering, K.N.Toosi University of Technology. His current research interests are Information Security, Wireless Networks, and Next Generation Networks.