Adaptive Access Control Scheme Utilizing Context Awareness in Pervasive Computing Environments

Jung Hwan Choi, Dong Hyun Kang, Hyunsu Jang, and Young Ik Eom
School of Information and Communication Engineering, Sungkyunkwan University, Korea
{themars, kkangsu, jhs4071, yieom}@ece.skku.ac.kr

Abstract

In pervasive computing environments, where various types of information are publicly owned, and multiple users access the networks via various networked devices anytime and anywhere, access control that grants permission to an authorized user is definitely needed for secure information access. Context awareness refers to the idea that computers can both sense and react based on various context in their environments. In many access control schemes, recently, context awareness has been utilized to guarantee dynamic access control according to current context and various access control schemes utilizing context awareness have been proposed. However, previous studies have difficulty describing conditions for assigning roles and modifying permissions. They also simply consider assigning roles or modifying permissions, rather than providing detailed access control algorithms such as role delegation or role revocation. In this paper, we propose an adaptive access control scheme utilizing context awareness in pervasive computing environments. We design an adaptive access control model based on traditional RBAC(Role-based Access Control) model, and present an adaptive access control scheme to guarantee dynamic user and permission assignment according to changes of context. In this scheme, we define context requirements in each table, enabling a more detailed description. We also guarantee dynamic access control via various access control algorithms.

1. Introduction

With the emergence of the pervasive computing era, multiple users are connecting networks without the interference of space and time, and various types of information and resources are publicly owned. In these environments, access control, which grants access permissions to an authorized user, has become an important factor [1]. Among access control policies, role-based access control(RBAC), which provides access permissions to roles rather than users, is a widely used access control model, because of its flexibility and efficiency [2].

In pervasive computing environments, where context information around the environments are collected in real-time and changed dynamically, utilization of various types of context information is definitely needed for dynamic access control [3]. Thus, roles of users and permissions of roles must be dynamically changed according to changes of context. Accordingly, many kinds of studies about access control schemes for pervasive computing environments have been performed, to date [4-11].

Y.G.Kim proposed a context aware access control mechanism for ubiquitous applications, which utilizes SCM(State Checking Matrix) to assign roles to users based on the current context [4]. G. Zhang suggested Dynamic RBAC(DRBAC), which changes roles or permissions based on the context via utilization of state machines [5]. Both of these schemes provide support for assigning roles or modifying permissions based on the current context. However, they have difficulty describing the constraints in detail, which is needed to construct user assignment(UA) and permission assignment(PA) according to the current context. Also, they fail to provide various types of access control algorithms such as role delegation or role revocation.

Our goal is to design an adaptive access control for pervasive computing environments, which guarantees dynamic UA and PA according to changes of context. Moreover, we provide a more detailed description method for context requirements and support for various access control algorithms via our adaptive access control scheme.

In this paper, we propose an adaptive access control scheme in pervasive computing environments. Our scheme functionally defines context requirements for dynamic UA and PA in each table, enabling a more detailed description. We provide various access control algorithms including role assignment, role delegation,

978-1-4244-3367-4/08/$25.00 ©2008 IEEE

role revocation, permission modification, and permission restoration.

The reminder of this paper is organized as follows: In Section 2, we examine the previous research relevant to this study. Section 3 introduces our proposed adaptive access control model and architecture, and Section 4 describes the adaptive access control algorithms. Then, in Section 5, we examine the trustworthiness of our proposed scheme. Finally, in Section 6, we present our conclusion.

2. Related work

Many kinds of studies about access control schemes utilizing context awareness have been studied to date, to grant or deny roles based on changes of context. In this section, we describe two access control schemes using context awareness.

2.1. Context-aware access control mechanism for ubiquitous applications

In 2003, Y. G. Kim designed the context-aware access control mechanism for ubiquitous applications [4]. This mechanism uses a state checking matrix(SCM) to grant or deny access privileges based on the current context. This mechanism consists of traditional RBAC and three important components: state-checking agent, SCM and context-aware agent. The state-checking agent maintains the role subset for each user. It monitors the current context of the users and dynamically changes the role activation of the user. SCM deals with context information such time, location, and resources. Therefore, it activates or deactivates specific roles of the user. The context-aware agent maintains the permission subset for each role. It plays a role in monitoring changes of the state checking matrix and dynamically changing the defaults of UA and PA to context-aware UA and PA at the time that SCM decides the activity level of the role for the user.

	$Location_1$	$Location_2$	$Location_3$	$Location_4$
$User(R_1)$	Active	Inactive	Inactive	Inactive
$Admin(R_2)$	Active	Active	Inactive	Active
$Pub(R_3)$	Inactive	Active	Active	Inactive

	$Time_1$	$Time_2$	$Time_3$	$Time_4$
$User(R_1)$	Active	Active	Inactive	Inactive
$Admin(R_2)$	Active	Active	Active	Active
$Pub(R_3)$	Inactive	Active	Active	Inactive

Figure 1. Example of state checking matrix

Figure 1 shows an example of SCM. SCMs for each context contain two values: active and inactive, which determine whether the role is activated or not. If all values in each SCM are active, the role is activated. In Figure 1, R_2 of $Location_2$ and R_2 of $Time_4$ are active, enabling the user to assume R_2. Via SCM, this mechanism dynamically adjusts UA and PA based on context information about the user.

2.2. Dynamic RBAC (DRBAC)

In 2003, Dynamic RBAC(DRBAC), which changes access privileges via a state machine, was proposed by G. Zhang [5]. This model provides support for dynamic, seamless and secure interactions between participating entities. Furthermore, in a highly dynamic and heterogeneous environment, the access privileges of an entity depend on its credentials, context and current state, which are dynamic.

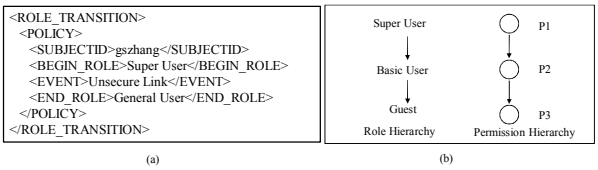

Figure 2. Examples of (a) role transition policy and (b) role and permission hierarchy

DRBAC utilizes two state machines: a role state machine for dynamic role assignment, a permission state machine for dynamic permission assignment. Roles and permissions are hierarchically constructed and have policies for transitions. Figure 2 shows examples of a role transition policy, and role and permission hierarchies. In this example, the role is changed from super-user to general-user, when the subject "*gszhang"* encounters the event "*Unsecure Link"* via the role transition policy. Permission transition is implemented in the same manner.

The aforementioned two mechanisms proposed by Y. G. Kim and G. Zhang have a similar goal, which is modification of traditional UA and PA via utilization of context awareness. However, in these two mechanisms, the description is inefficient for expressing constraints for UA and PA. In Y.G.Kim's scheme, multiple SCMs are needed depending on the number of contexts for expressing requirements. If it needs modification for a requirement, all SCMs must be modified. In G. Zhang's scheme, every event policy for a transition must be defined in XML format. Moreover, they do not support detailed access control mechanisms. Thus, neither scheme provides various access control mechanisms including role delegation, role revocation, and permission modification.

3. Adaptive access control

In this section, we present the adaptive access control model, which dynamically grants roles or modifies permissions based on traditional RBAC via utilization of context awareness. We formalize each component of the adaptive access control model via definitions, to explain our scheme in detail and describe the adaptive access control architecture.

3.1. Adaptive access control model

Adaptive access control guarantees dynamic UA and PA according to the current context, via utilization of context aware data collected from pervasive computing environments.

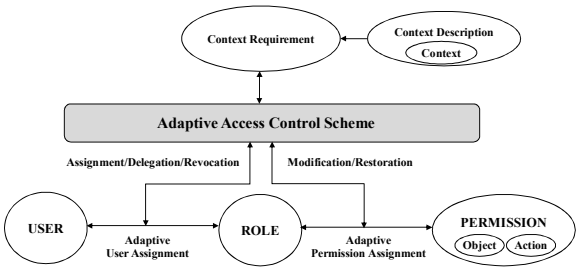

Figure 3. Adaptive access control model

Figure 3 illustrates our proposed adaptive access control model. The adaptive access control model is based on a traditional RBAC model devised by NIST [12]. User, role, and permission are the original components of a traditional RBAC. In a traditional RBAC model, UA and PA are handled by administrators; the adaptive access control model, however, provides adaptive UA and adaptive PA based on context requirements via the adaptive access control scheme. Thus, adaptive access control sets context requirements and provides adaptive UA and adaptive PA dynamically depending on the satisfaction of context requirements. Adaptive access control is suitable for pervasive computing environments, since this model grants or denies roles, and modifies or restores permissions automatically, without the intervention of administrators. A context description, which depicts current contexts in detail, is needed to describe context requirements.

3.2. Component formalization

Each component of the adaptive access control model consists of basic RBAC components and extended components that utilize context awareness. Basic RBAC components are composed of user, role and permission. A user is an entity whose access is controlled via assigned roles. Each user possesses various roles and each role can also be assigned to multiple users, enabling them to have multiple-to-multiple relationships. A role is a set of related permissions. Each role has multiple permissions, and each permission can be assigned to various roles. They also have many-to-many relationships. Permission is approval to access or operate resources and consists of an object and an action, described as <o, a> where *o* is an object, and *a* is an action.

Now, we formalize extended components for adaptive UA and PA as follows. Extended components are composed of context, context description, context requirement, user assignment and permission assignment.

Definition 1. (Context) Let C be the set of all contexts, the context $C_i \in C$. C_i = <contextName, contextAttr> and ContextAttr = <attrName, attrType, attrValue>. Thus, C_i = <contextName, (attrName, attrType, attrValue)>.

Context represents measurable state information, such as the location of a user, temperature of the room, and current time. In our model, the context comprises the context name and context attribute. The context attribute comprises the attribute name, attribute type and attribute value. The context name includes the time and location constituting the context. The context attribute includes more detailed information about the context. Example 1 and 2 show how we define location and time context.

Ex1) $Location_1$ = *<Location, (Laboratory, String, Distributed Computing)>*

Ex2) $Time_1$ = *<Time, (Morning, Integer, 0900)>*

Definition 2. (Context Description) Let CD be the set of context descriptions, the context description $CD_i \in$ CD. CD_i = <subjectID, $(C_1 \wedge C_2 \wedge ... \wedge C_n)$>.

The context description consists of various contexts, and the subject that handles these contexts. CD represents the current states in detail. The subject can be a user or an object. In example 3, CD_1 depicts the state in which Bob is located at $location_1$ and $time_1$.

Ex3) CD_1 = *<Bob,* $(C_{location1} \wedge C_{time1})$*>*

Definition 3. (Context Requirement) Let CR be the set of context requirements, $CR_i \in$ CR. CR_i = <CRE_1, CRE_2, ..., CRE_n> where CRE is the context requirement elements. CRE_i = <CD_i, positiveOrNegative> and it returns true or false.

Context requirement(CR) is a requirement for providing adaptive UA and adaptive PA according to the current context. UA and PA are provided when context requirements are satisfied. Thus, CR is set to provide dynamic UA and PA depending on its satisfaction. CR consists of context requirement elements(CREs), which include the context description and its satisfaction status. Under current context, CRE returns true or false, depending on the satisfaction of CD with a *positiveOrNegative* state. If all CREs return true, CR returns true. Otherwise, CR returns false.

Assignment is needed for linking user-to-role and role-to-permission relationships. These are called user assignment(UA) and permission assignment(PA), respectively. For dynamic access control, we formally define UA and PA as follows:

Definition 4. (User Assignment) Let UAT be the user assignment table, $UAE_i \in UAT$, where UAE is a user assignment element. $UAE_i = <U_j , R_k ,$ delegatationStatus>

User assignment represents a link between a user and an assigned role. A user assignment element(UAE), which is a component of the user assignment table(UAT), contains a user, an assigned role and the delegated status of the role. UAE is stored in UAT and represents the current user assignment state.

Definition 5. (Permission Assignment) Let PAT be the user assignment table, $PAE_i \in PAT$, where PAE is a permission assignment element. $PAE_i = <R_j , P_k>$

Permission assignment represents a link between a role and an assigned permission. A permission assignment element(PAE), which is a component of permission assignment table(PAT), contains a role and an assigned permission and is stored in PAT. In the same manner, PAT represents the current permission assignment state.

3.3. Adaptive access control architecture

Figure 4 shows the architecture of the adaptive access control scheme to provide adaptive UA and PA to users. Adaptive access control is mainly operated via the access control manager(ACM), which controls the processing of access control requests and transfer of updated context information. And, the other two managers, the adaptive user assignment manager(AUAM) and the adaptive permission assignment manager(APAM), implement access control utilizing context awareness. AUAM provides role assignment, role delegation, and role revocation based on context requirements defined in each table. APAM supports permission modification and restoration based on context requirements.

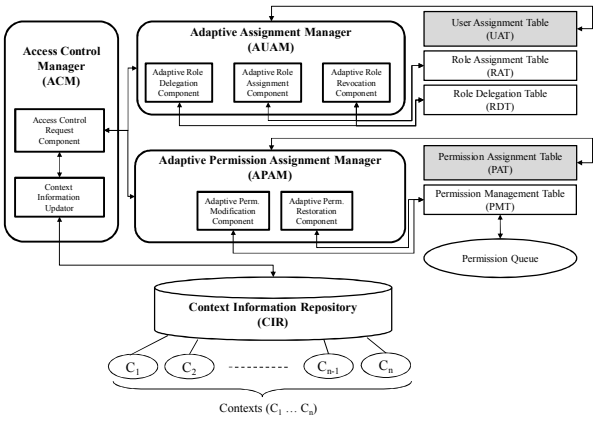

Figure 4. Adaptive access control architecture

The context information repository(CIR) contains context information(C_1, C_2, …, C_n) collected from various sensors and the user's multi-modalities in pervasive computing environments. However, this area is beyond the scope of our paper, thus, we do not consider it here. We assume that context information is automatically collected in CIR.

4. Adaptive access control scheme

In this section, we introduce the adaptive access control scheme based on our proposed adaptive access control model. We present various access control algorithms for providing adaptive UA and PA via utilization of context awareness.

4.1. Adaptive user assignment (AUA)

Adaptive user assignment(AUA) means assigning a role to a user according to the current context. AUA is implemented via AUAM and conducts dynamic user assignment based on predefined context requirements, without the intervention of administrators. There are three functions for AUA: adaptive role assignment, adaptive role delegation, and adaptive role revocation.

4.1.1. Adaptive role assignment. Adaptive role assignment automatically grants a role to a user according to the current context. Context requirements must be set for adaptive role assignment in the role assignment table(RAT). In RAT, data is stored in the form of role assignment elements(RAEs). A RAE is defined as follows:

$RAE_i \in RAT$, where $RAE_i = <R_j, CR_{RA}>$

R_j means a role that will be assigned, and CR_{RA} represents a point in time at which a role is assigned. When CR_{RA} returns true via satisfaction of all CREs in CR_{RA}, the role, R_j, is included in UAT, to assign a role to a user. Table 1 shows the procedure for adaptive role assignment.

Table 1. Adaptive role assignment algorithm

Algorithm 1. Adaptive Role Assignment

```
INPUT User U, RAE(1..n) ∈ RAT, Current context C
OUTPUT UAEz
BEGIN
    WHILE RAE.CR != true
        Set result to 0
        Read current context C
        FOR i=0 to n DO
            IF RAE.CR.CREi = true THEN result ← result + 1
            END IF
        END FOR
        IF result == n THEN RAE.CR ← true
        ELSE THEN RAE.CR ← false
        END IF
    END WHILE
    Create UAEz with <U, RAE.R, none>
    Insert UAEz to UAT
END
```

4.1.2. Adaptive role delegation. Adaptive role delegation automatically delegates all roles of a certain user to a specified user, according to the current context. Context requirements must be set for adaptive role delegation in the role delegation table(RDT). In RDT, data is stored in the form of role delegation elements(RDEs). A RDE is defined as follows:

$RDE_i \in RDT$ with $RDE_i = <CR_{RD}, U_{delegator}, U_{delegatee}>$

Table 2. Adaptive role delegation algorithm

Algorithm 2. Adaptive Role Delegation

```
INPUT User U, RDE(1..n) ∈ RDT, Current context C
OUTPUT UAEz
BEGIN
    WHILE RDE.CR != true
        Set result to 0
        Read current context C
        FOR i=0 to n DO
            IF RDE.CR.CREi = true THEN result ← result + 1
            END IF
        END FOR
        IF result == n THEN RDE.CR ← true
        ELSE THEN RDE.CR ← false
        END IF
    END WHILE
    FOR z=0 to n DO
    Create UAEz with <Udelegatee, UdelegatorRz, Udelegator>
    Insert UAEz to UAT
    END FOR
END
```

CR_{RD} represents a point in time at which a role is assigned. $U_{delegator}$ and $U_{delegatee}$ mean the user who delegates roles and the user to which roles are assigned, respectively. When CR_{RD} returns true via satisfaction of all CREs in CR_{RD}, all roles of $U_{delegator}$ are delegated to $U_{delegatee}$ via inclusion of the new UAEs in UAT. Table 2 is the pseudo-code for adaptive role delegation.

4.1.3. Adaptive role revocation. Adaptive role revocation immediately denies roles that were assigned or delegated previously, when context requirements are not satisfied. Table 3 is the procedure for adaptive role revocation.

Table 3. Adaptive role revocation algorithm

Algorithm 3. Adaptive Role Revocation

```
INPUT User U, assigned RAE(1..n) ∈ RAT, assigned RDE(1..m) ∈ RDT, Current context C
OUTPUT UAEz
BEGIN
    WHILE RAE.CR == true || RDE.CR == true
        Set result to 0
        Set result2 to 0
        Read current context C
        FOR i=0 to n DO
            IF RAE.CR.CREi = true THEN result ← result + 1
            END IF
        END FOR
        FOR i=0 to m DO
            IF RDE.CR.CDEi = true THEN result2 ← result2 + 1
            END IF
        END FOR
        IF result == n THEN RAE.CR ← true
        ELSE IF result2 == m THEN RDE.CR ← true
        ELSE THEN RAE.CR ← false, RDE.CR ← false
        END IF
    END WHILE
    IF RAEi.CR == false THEN Delete UAEz from UAT where UAEz is <U, RAEi.R, none>
    ELSE IF RDEi.CR == false THEN
        FOR z=0 to n DO
            Delete UAEz from UAT where UAEz.Udelegator == RDEi.Udelegator
        END FOR
    END IF
END
```

4.2. Adaptive permission assignment (APA)

Adaptive permission assignment(APA) means modification of a permission of a role according to current context information. APA is implemented via APAM and conducts dynamic permission assignment based on predefined context requirements in the same manner as AUA. There are two functions for APA: adaptive permission modification and adaptive permission restoration.

4.2.1. Adaptive permission modification. Adaptive permission modification automatically modifies a permission of a role according to the current context. Context requirements must be set to adaptive permission modification in the permission management table(PMT). In PMT, sources are stored in the form of permission management elements(PMEs). A PME is defined as follows:

$PME_i \in PMT$, where $PME_i = <R_j, P_k, CR_{PM}, Condition>$

R_j and P_k represent a role and an assigned permission, respectively. CR_{PM} represents a point in time at which permissions are modified. Condition represents a state that will be changed. Condition can be *disable* for making P_k to deactivate or other actions such as *read* or *write* for the permission P_k. When CR_{PM} returns true via satisfaction of all CREs in CR_{PM}, the action of the permission P_k is modified to the specified condition. If

the condition is described as *disable*, it stores the original permission P_k in the permission queue and deletes it from PAT. Otherwise, it stores the original permission P_k in the permission queue, modifies its action to the specified condition, and updates it in PAT. Table 4 is the pseudo-code for adaptive permission modification.

Table 4. Adaptive permission modification algorithm

Algorithm 4. Adaptive Permission Modification

```
INPUT RME(1...n) in RMT, PAE(1...m) ∈ PAT, Current context C, PermissionQueue
OUTPUT PAEz
BEGIN
    WHILE PME.CR != true
        Set result to 0
        Read current context C
        FOR i=0 to n DO
            IF PME.CR.CREi = true THEN result ← result + 1
            END IF
        END FOR
        IF result == n THEN PME.CR ← true
        ELSE THEN RDE.CR ← false
        END IF
    END WHILE
    IF PME.Condition == disable THEN
        Copy PAEz to PermissionQueue where PAEz = <PME.R, PME.P>
        Delete PAEz from PAT
    ELSE THEN
        Copy PAEz to PermissionQueue where PAEz = <PME.R, PME.P>
        Modify PAEz.P.A to PME.Condition
        Update PAEz to PAT
    END IF
END
```

4.2.2. Adaptive permission restoration. Adaptive permission restoration automatically restores a permission of a role, which was previously modified, to its past condition, according to the current context.

Table 5. Adaptive permission restoration algorithm

Algorithm 5. Adaptive Permission Restoration

```
INPUT assigned RME(1...n) ∈ RMT, PAE(1...m) ∈ PAT, Current context C, PermissionQueue
OUTPUT PAEz
BEGIN
    WHILE PME.CR == true
        Set result to 0
        Read current context C
        FOR i=0 to n DO
            IF PME.CR.CREi = true THEN result ← result + 1
            END IF
        END FOR
        IF result == n THEN PME.CR ← true
        ELSE THEN RDE.CR ← false
        END IF
    END WHILE
    IF PME.Condition == disable THEN
        Get PAEz from PermissionQueue where PAEz.P == PME.P
        Insert PAEz to PAT
    ELSE THEN
        Get PAEz from PAT where PAEz.P == PME.P
        Modify PAEz.P.A to PME.Condition
        Update PAEz to PAT
    END IF
END
```

Adaptive permission restoration proceeds in two different ways, depending on the past condition. If the past condition is described as *disable*, it includes the past permission in PAT for restoration. Otherwise, it modifies the current condition to the past condition, and updates the modified permission in PAT. Table 5 is the pseudo-code for adaptive permission restoration.

5. Trustworthiness analysis

In this section, we analyze the trustworthiness of our proposed adaptive access control scheme. We evaluate whether our proposed scheme guarantees adaptive access control based on current context via theorems and proofs. Table 7 describes our notations for evaluations.

Table 7. Notations for evaluations

Notations	Descriptions
getUpdatedContext()	Fetching current context information from *CIR*
getTableInfo(tableName)	Fetching table information
getQueueInfo(P_i)	Fetching permission P_i from permission queue
adaptiveRoleAssign(RAE_i)	Assigning $RAE_i.R_j$ to U_i described in RAE_i
adaptiveRoleDelegate(RDE_i)	Delegating all roles of $RAE_i.U_{delegator}$ to $RAE_i.U_{delegatee}$
adaptiveRoleRevoke()	Revoking assigned roles or delegated roles
checkCR()	Check whether current *CIR* is true or false

Prior to the analysis, we assume that context requirements for adaptive access control are set to each table as follows:

Table 8. Examples of context requirements

Table Name	Elements
Role Assignment Table (RAT)	CRE_1 = < (Scheduler, < (Bob, (schedule, String, presentation)) ∧ (Time, (afternoon, Integer, 0300)) ∧ (Location, (office, String, room A)) >), positive > CRE_2 = < (Bob, < (Time, (afternoon, Integer, 0300)) ∧ (Location, (office, String, room A)) >), positive > CR_{RA} = < CRE_1, CRE_2 > RAE = <$R_{presenter}$, CR_{RA} >
Role Delegation Table (RDT)	CRE_1 = < (Scheduler, < (Time, (day, Integer, 20081001)) ∨ (Time, (day, Integer, 20081005)) ∧ (Bob, (schedule, String, businesstrip)) >), positive > CRE_2 = < (Bob, < (Time, (day, Integer, 20081001)) ∨ (Time, (day, Integer, 20081005)) >), positive > CRE_3 = < (Bob, < (Location, (office, String, Bob's room)>), negative > CR_{RD} = < CRE_1, CRE_2, CRE_3 > RDE = < CR_{RD}, U_{Bob}, U_{John} >
Permission Management Table (PMT)	CRE_1 = < (Projectsystem, <(Bob, (member, String, debugger)) ∨ (John, (member, String, projectmanager))>), positive > CRE_2 = < (Bob, <(Location, (building, String, R4))>), positive > CRE_3 = < (John, <(Location, (building, String, R4))>), positive > CR_{PM} = < CRE_1, CRE_2, CRE_3 > PME = <$R_{nurseRole}$, $P_{accessData}$, CR_{PM}, read>

Table 8 shows examples of the context requirements described in each table. In RAT, the context requirement for assigning a presenter role to Bob is described when presentation is scheduled to Bob and Bob meets those conditions. In RDT, the context requirement is defined for delegating all of Bob's roles to John, when Bob goes on a business trip. The context requirement is specified for modifying the access permission in PMT. According to the described context requirement, Bob can access to the system server only when he is located in the building with a project manager.

(Theorem 1) The adaptive access control scheme guarantees adaptive role assignment, delegation, and revocation according to the current context.

Table 9. Proof for theorem 1

Proof :
$ACM \rightarrow CIR$: *getUpdatedContext()*
$CIR \rightarrow ACM$: *updatedContext*
$ACM \rightarrow AUAM$: *updatedContext*
AUAM : *checkCR()*
if (CR_{RA} == *true) then*
$AUAM \rightarrow UAT$: *adaptiveRoleAssign*(RAE_i)
UAT : $UAE = <U, RAE_i.R, none>$
else if (CR_{RD} == *true) then*
$AUAM \rightarrow UAT$: *adaptiveRoleDelegate(RDEi)*
UAT : $UAE_{(1...i)} = <RDE.U_{delegatee}, U_{delegatee}.R_{(1...i)}, RDE.U_{delegator}>$
else if (CR_{RA} == *false or* CR_{RD} == *false) then*
$AUAM \rightarrow UAT$: *adaptiveRoleRevoke()*
UAT : *delete*(UAE_i)

Table 9 shows the proof for theorem 1. Context requirements are checked in AUAM based on the currently updated context, and it is determined whether context requirements are satisfied or not. If CR_{RA} in RAT is satisfied for the current context, the role is assigned to a user referenced in RAE. In this manner, roles are delegated to a specific user based on RDE, when CR_{RD} in RDT is satisfied for the current context. Previously assigned or delegated roles can be immediately revoked, when CR_{RA} or CR_{RD} is not satisfied for the current context. Thus, this scheme guarantees adaptive role assignment, delegation, and revocation according to the current context.

(Theorem 2) The adaptive access control scheme guarantees adaptive modification of permissions and their restoration according to the current context.

Table 10. Proof for theorem 2

Proof :
$ACM \rightarrow CIR$: *getUpdatedContext()*
$CIR \rightarrow ACM$: *updatedContext*
$ACM \rightarrow APAM$: *updatedContext*
APAM : *checkCR()*
if (CR_{PM} == *true) then*
$APAM \rightarrow PAT$: *getTableInfo(PMEi)*
$PAT \rightarrow APAM$: PME_i
$APAM \rightarrow PQ$: *copy*($PME_i.P$)
$APAM \rightarrow PAT$: *update*(PME_i)
PAT : $PME_i.P = <$ *Object,* $PME_i.Condition >$
else if (CR_{PM} == *false) then*
$APAM \rightarrow PAT$: *getTableInfo*(PME_i)
$PAT \rightarrow APAM$: PME_i
$APAM \rightarrow PQ$: *getQueueInfo*($PME_i.P$)
$PQ \rightarrow APAM$: $PME_i.P'$
if (Compare($PME_i.P$, $PME_i.P'$ == *false)*
$APAM \rightarrow PAT$: *update*(PME_i)
PAT : $PME_i.P = <$ *Object,* $PME_i.P' >$

Table 10 shows the proof for theorem 2. Context requirements are checked in APAM based on the updated current context, and it is determined whether the context requirements are satisfied or not. If CR_{PM} in PMT is satisfied for the current context, the permission specified in PME is modified to the referenced permission. The previous permission is stored in the permission queue for restoration. For previously modified permissions, permission restoration is immediately performed, when CR_{PM} is not satisfied for the current context. Previously modified permissions can be determined via comparison of the current permission and stored permissions. Thus, this scheme guarantees adaptive modification of permissions and their restoration according to the current context.

6. Conclusion

In this paper, we proposed an adaptive access control scheme in pervasive computing environments. In our scheme, we provided adaptive UA and PA according to the current context based on the satisfaction of context requirements. Context requirements were defined in each table, enabling us to construct a more detailed description. We also guaranteed dynamic access control via various access control algorithms including role assignment, role delegation, role revocation, permission modification, and permission restoration.

7. Acknowledgement

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement) (IITA-2008-(C1090-0801-0046))

References

[1] X. Feng, X. Jum. H. Hao, and X. Li, "Context-Aware Role-Based Access Control Model for Web Services", *GCC 2004 Workshops*, LNCS #3252, Springer-Verlag, 2004, pp. 430-436.

[2] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control*, Artech House Publishers, 2003.

[3] D. Kulkarni and A. Tripathi, "Context-Aware Role-based Access Control in Pervasive Computing Systems", *ACM symposium on access control models and technologies(SACMAT'08)*, ACM, 2008, pp. 113-122.

[4] Y. G. Kim, C. J.Mon, D. Jeong, C. Y. Song, and D. K. Baik, "Context-Aware Access Control Mechanism for Ubiquitous Applications", *AWIC 2005*, LNAI #3528, Springer-Verlag, 2005, pp. 236-242.

[5] G. Zhang, "Dynamic Context Aware Access Control for Grid Applications", *a thesis for the degree of Master*, The state university of New Jersey, 2003, pp. 1-40.

[6] N. Gustaf and S. Mark, "An Approach to Engineer and Enforce Context Constraints in an RBAC Environment", *In 8th ACM Symposium on Access Control Models and Technologies (SACMAT2003)*, ACM, Como, Italy, 2003, pp. 65-79.

[7] K. I. Kim, H. J. Ko, H. S. Hwang, and U. M. Kim, "Context RBAC/MAC Access Control for Ubiquitous Environment", *Proc. of DASFAA 2007*, LNCS #4443, Springer-Verlag, 2007, pp. 1075-1085.

[8] W. Yao, K. Moody, and J. Bacon, "A Model of OASIS Role-Based Access Control and its Support for Active Security", *ACM symposium on access control models and technologies(SACMAT'01)*, ACM, 2001, pp. 171-181.

[9] X. Tang, Y. Zhang, and J. You, "RCACM : Role-Based Context Coordination Model for Mobile Agent Applications", *Proc. of the 2nd Int'l. Workshop on Grid and Cooperative Computing*, 2003, 702-705.

[10] M. J. Covington, W. Long, S. Srinivasan, A. K. Dey, M. Ahamad, and G. D. Abowd, "Securing Context-Aware Applications Using Environment Roles", *ACM symposium on access control models and technologies(SACMAT'01)*, ACM, 2001, pp. 10-20.

[11] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka, "Context-Based Secure Resource Access in Pervasive Computing Environments", *Proc. of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops(PERCOMW'04)*, IEEE, 2004, pp. 159-163.

[12] H. Feinstein R. Sandhu, E. Coyne and C. Youman, "Role-based access control models", *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 38-47.