JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 26, 1801-1816 (2010)

Short Paper

CA-RBAC: Context Aware RBAC Scheme in Ubiquitous Computing Environments*

JUNG HWAN CHOI, HYUNSU JANG AND YOUNG IK EOM+
School of Information and Communication Engineering
Sungkyunkwan University
Suwon, 440-746 Korea

Role based access control (RBAC) assigns access permissions to a role rather than a user. This simplifies access control management by simply assigning appropriate roles to users and by modifying the permissions of the roles. With the advent of ubiquitous computing, many kinds of services, especially personalized services to give convenience to users, have been introduced. Among them, providing proper access permissions to users based on the current context has become an important issue because their status and access privilege should be dynamically changed. Previously, various access control methods utilizing context awareness have been proposed; however, their constraint description methods, used to assign a role to a user, are quite complex and not enough to express detailed context. Also, they do not fully cover the various situations that can be occurred in ubiquitous computing environments. In this paper, we propose an access control scheme, combining RBAC with context awareness, to give proper privilege to users based on their current context in ubiquitous computing environments. Our scheme defines the constraints for assigning a role and modifying the permissions of each role, enabling more detailed descriptions. We also provide various access control algorithms to support diverse situations which occur in ubiquitous computing environments.

Keywords: role based access control, context awareness, ubiquitous computing, context based access control, personalized access control

1. INTRODUCTION

With the emergence of the ubiquitous computing era, multiple users utilize combined services, and various types of resources and information are publicly owned. In these environments, access control, which grants access permissions to an authorized user, has become an essential factor [1]. Among access control policies, role-based access control (RBAC), which provides access permissions to roles rather than users, is an access control model that is widely used, because of its flexibility and efficiency [2].

In ubiquitous computing environments, where the state of users and context information are collected in real-time and changed dynamically, utilization of various types of context information for providing roles or modifying permissions 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. Therefore, many kinds of studies

Received October 27, 2008; revised June 11, 2009; accepted August 28, 2009.
Communicated by Pau-Choo Chung.
* This research was supported by MKE, Korea under ITRC NIPA-2010-(C1090-1021-0008).
+ Corresponding author.

about access control mechanisms for ubiquitous 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, 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 mechanisms such as role delegation or personalized access control that considers the user's preferences because they do not fully consider the various situation that can be occurred in ubiquitous computing environments.

In this paper, we propose a context aware RBAC (CA-RBAC) scheme in ubiquitous 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 mechanisms including role assignment, role delegation, role revocation, permission modification, and permission restoration. We also guarantee personalized access control that considers the user's preferences.

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

2. RELATED WORK

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

2.1 Context-Aware Access Control Mechanism for Ubiquitous Applications

In 2003, context-aware access control mechanism for ubiquitous applications was designed by Y. G. Kim [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. SCM deals with context information such location, time, and resources (*i.e.*, network bandwidth and memory usage). The context-aware agent maintains the permission subset for each role. It monitors changes of the state checking matrix and dynamically changes the defaults of PA and UA to context-aware PA and UA at the time that SCM decides the activity level of the role for the user.

Fig. 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. When all values ineach SCM turn active, then the role will be activated. In Fig. 1, the user locates in

	$Location_1$	$Location_2$	$Location_3$	$Location_4$
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

Fig. 1. Example of the state checking matrix (SCM).

$Location_2$ at $Time_4$, therefore, the user will get R_2, because all values are active. Via SCM, it dynamically adjusts UA and PA based on context information about the user.

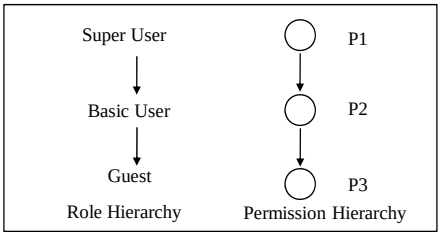
2.2 Dynamic RBAC (DRBAC)

In 2003, G. Zhang designed Dynamic RBAC (DRBAC), which changes access privileges via a state machine [5]. This model provides support for dynamic, seamless and secure interactions between participating entities.

DRBAC utilizes two state machines: role state machine for dynamic role assignment, permission state machine for dynamic permission assignment. Roles and permissions are hierarchically constructed and have policies for transition. Fig. 2 shows examples of a role transition policy and role and permission hierarchy. Role transition policy in Fig. 2 defines the role change with XML format. Based on this policy, 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.

```
<ROLE_TRANSITION>
  <POLICY>
    <SUBJECTID>gszhang</SUBJECTID>
    <BEGIN_ROLE>Super User</BEGIN_ROLE>
    <EVENT>Unsecure Link</EVENT>
    <END_ROLE>General User</END_ROLE>
  </POLICY>
</ROLE_TRANSITION>
```

(a)



(b)

Fig. 2. Examples of (a) Role transition policy and (b) Role and permission hierarchy.

3. DESIGN OF CA-RBAC

In this section, we present the CA-RBAC model, which dynamically grants roles or denies permissions based on traditional RBAC via utilization of context awareness. We also formalize each component of the CA-RBAC model via definitions, to explain our scheme in detail.

3.1 CA-RBAC Model

CA-RBAC guarantees dynamic user assignment (UA) and permission assignment (PA) according to the current context, via utilization of context awareness data collected from ubiquitous computing environments. Fig. 3 illustrates our proposed CA-RBAC model. The CA-RBAC scheme is adapted from 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 CA-RBAC model, however, provides UA and PA based on context requirements via the CA-RBAC scheme. Thus, CA-RBAC sets context requirements and provides dynamic UA and PA depending on the satisfaction of context requirements. CA-RBAC is suitable for ubiquitous 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.

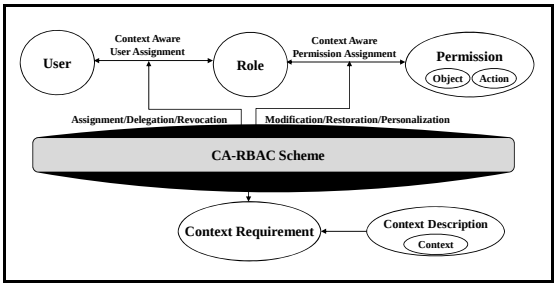


Fig. 3. CA-RBAC model.

3.2 Component Formalization

Each component of the CA-RBAC model consists of basic RBAC components and extended components that utilize context awareness. We formalize these components as follows. First, we define the basic RBAC components: user, role, permission, user assignment and permission assignment.

Definition 1 (User) Let U be the set of users, the user $u_i \in U$. Various roles (r_1, r_2, …, r_m) are assigned to each user (u_1, u_2, …, u_n).

Definition 2 (Role) Let R be the set of roles, the role $ri \in R$. Each role (r_1, r_2, …, r_m) has various permissions (p_1, p_2, …, p_n).

Definition 3 (Permission) Let P be the set of permissions, the permission $p_i \in P$. p_i = <O, A> where O is an object, and A is an action.

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 also 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.

Assignment is needed for linking user-to-role and role-to-permission relationships. These are called user assignment and permission assignment, respectively. They are formally defined as follows:

Definition 4 (User Assignment) Let UAT be the user assignment table, $UAE_i \in UAT$, where UAE is the user assignment element. $UAE_i = <u_j, r_k, \text{delegatationStatus}>$

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

A user assignment (UA) represents a link between a user and an assigned role. A user assignment element (UAE), 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 states. Permission assignment (PA) represents a link between a role and an assigned permission. Permission assignment element (PAE), 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.

Next, we define the extended RBAC components, which utilize context awareness to provide dynamic access control: context, context description, and context requirement.

Definition 6 (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. For example, we define location and time context as follows:

Ex. 1: Location_1 = <Location, (Laboratory, String, Distributed Computing)>
Ex. 2: Time_1 = <Time, (Morning, Integer, 0900)>

Definition 7 (Context Description) Let CD be the set of context descriptions, the context description $CD_i \in CD$. CD_i = <subjectID, $(C_1 \wedge C_2 \wedge \ldots \wedge C_n)$>.

The context description consists of various contexts, and the subject that handles these contexts and represents the current states in detail. The subject can be a user or an object. We depict the state in which Bob is located at location_1, at time_1, as follows:

Ex. 3: CD_1 = <Bob, $(C_{\text{location1}} \wedge C_{\text{time1}})$>

Definition 8 (Context Requirement) Let CR be the set of context requirements, $CR_i \in CR$. $CR_i = <CRE_1, CRE_2, \ldots, CRE_n>$ where CRE is the context requirement elements. $CRE_i = <CD_i, \text{condition}>$ and it returns true or false.

Context requirement (CR) is a requirement for providing dynamic UA and 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 (CRE), 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 condition state. There are two conditions; one is positive and the other is negative. If the condition is set to positive, it returns true when CD are satisfied. Otherwise, if the condition is set to negative, it returns false when CD are satisfied. If all CREs return true, CR returns true. Otherwise, CR returns false.

4. CA-RBAC SCHEME

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

4.1 Architecture

Fig. 4 shows the architecture of the CA-RBAC scheme. The CA-RBAC scheme 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 context aware user assignment manager (CAUAM) and context aware permission assignment (CAPAM), implement access control utilizing context awareness. CAUAM provides role assignment, role delegation, and role revocation based on context requirements defined in each table. CAPAM not only supports permission modification and restoration based on context requirements, but also provides personalized access control via utilization of user preference information in the user profile repository (UPR).

Regarding the two repositories, the context information repository (CIR) contains context information (C_1, C_2, …, C_n) collected from various sensors and the user's multimodalities. UPR includes the status and preference information of the users (U_1, U_2, …, U_n). However, this area is beyond the scope of our paper, thus, we do not consider it here. We assume that context information and user profiles are automatically collected in each repository.

4.2 Context Aware User Assignment (CAUA)

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

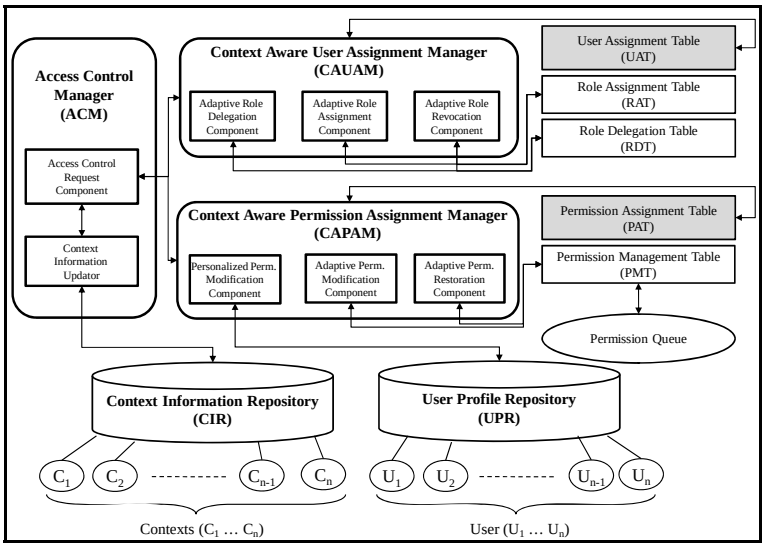


Fig. 4. CA-RBAC architecture.

4.2.1 Adaptive role assignment

Adaptive role assignment automatically grants a role to a user according to the current context. Context requirements must be set for dynamic role assignment in the role assignment table (RAT). In RAT, data is stored in the form of role assignment elements (RAEs). An 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 condition to assign a role when it returns true. Table 1 shows the procedure for adaptive role assignment. 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.

4.2.2 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 dynamic role delegation in the role delegation table (RDT).

In RDT, data is stored in the form of role delegation elements (RDEs). An RDE is defined as follows:

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

CR_{RD} represents a condition to delegate a role when it returns true. $u_{delegator}$ and $u_{delegatee}$ mean the user who delegates roles and the user to which roles are assigned, respectively. Table 2 represents the pseudo-code for adaptive role delegation. 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 1. Adaptive role assignment algorithm.

Input	User U, $RAE_{(1...n)} \in$ RAT, Current context C
Output	UAE_z
Procedure	**begin** **while** RAE.CR! = true set result to 0 read current context C **for** i = 0 to n **do** **if** RAE.CR.CRE_i = true **then** result ← result + 1 **end if** **end for** /* CR is true when all CREs are true */ **if** result == n **then** RAE.CR ← true **else** RAE.CR ← false **end if** **end while** /* assign a role by inserting UAE to UAT */ create UAE_z with <u, RAE.r, none> insert UAE_z to UAT **end**

Table 2. Adaptive role delegation algorithm.

Input	User U, $RDE_{(1...n)} \in$ RDT, Current context C
Output	UAE_z
Procedure	**begin** **while** RDE.CR! = true set result to 0 **r**ead current context C **for** i = 0 to n **do** **if** RDE.CR.CRE_i = true **then** result ← result + 1 **end if** **end for** /* CR is true when all CREs are true */ **if** result == n **then** RDE.CR ← true **else** RDE.CR ← false **end if** **end while** /* delegate the roles by inserting UAE to UAT with delegator field*/ **for** z = 0 to n **do** create UAE_z with <$u_{delegatee}$, $u_{delegator}.r_z$, $u_{delegator}$> insert UAE_z to UAT **end for** **end**

4.2.3 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. When CR_{RA} or CR_{RD} returns false via dissatisfaction of any CREs in CR_{RA} or CR_{RD}, respectively, the role is immediately denied via removal of the UAEs in UAT.

Table 3. Adaptive role revocation algorithm.

Input	User U, assigned $RAE_{(1...n)} \in$ RAT and $RDE_{(1...m)} \in$ RDT, Current context C
Output	UAE_z
Procedure	**begin** **while** RAE.CR == true || RDE.CR == true set result to 0 set result 2 to 0 read current context C **for** i = 0 to n **do** **if** $RAE.CR.CRE_i$ = true **then** result ← result + 1 **end if** **end for** **for** i = 0 to m **do** **if** $RDE.CR.CDE_i$ = true **then** result 2 ← result 2 + 1 **end if** **end for** **if** result == n **then** RAE.CR ← true **else if** result 2 == m **then** RDE.CR ← true **else** RAE.CR ← false, RDE.CR ← false **end if** **end while** /* when UAEs for assigned roles or delegated roles are not satisfied, assignment and delegation are canceled by removing UAE from UAT */ **if** RAE_i.CR == false **then** delete UAE_z from UAT where UAE_z is <u, $RAE_i.r$, none> **else if** RDE_i.CR == false **then** **for** z = 0 to n **do** delete UAE_z from UAT where $UAE_z.u_{\text{delegator}}$ == $RDE_i.u_{\text{delegator}}$ **end for** **end if** **end**

4.3 Context Aware Permission Assignment (CAPA)

Context aware permission assignment (CAPA) means modification of a permission of a role according to current context information. CAPA is implemented via CAPAM and conducts dynamic permission assignment based on predefined context requirements in the same manner as CAUA. There are three functions of CAPA: adaptive permission modification, adaptive permission restoration, and personalized permission modification.

4.3.1 Adaptive permission modification

Adaptive permission modification automatically modifies a permission of a role according to the current context. Context requirements must be set to dynamic 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 condition to modify a permission when it returns true. 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.

Table 4 describes the pseudo-code for adaptive permission modification. 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 *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. Adaptive permission modification algorithm.

Input	$PME_{(1...n)}$ in PMT, $PAE_{(1...m)} \in$ PAT, Current context *C*, PermissionQueue
Output	PAE_z
Procedure	**begin** **while** PME.CR! = true set result to 0 read current context *C* **for** *i* = 0 to *n* **do** **if** PME.CR.CRE_i = true **then** result ← result + 1 **end if** **end for** /* CR is true when all CREs are true */ **if** result == *n* **then** PME.CR ← true **else** RDE.CR ← false **end if** **end while** /* modify the permission by updating PAE to PAT */ **if** PME.Condition == disable **then** copy PAE_z to PermissionQueue where PAE_z = <PME.*r*, PME.*p*> delete PAE_z from PAT **else** copy PAE_z to PermissionQueue where PAE_z = <PME.*r*, PME.*p*> modify $PAE_z.p.A$ to PME.Condition update PAE_z to PAT **end if** **end**

4.3.2 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 represents the pseudo-code for adaptive permission restoration. 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 enable. Otherwise, it modifies the current condition to the past condition, and updates the modified permission in PAT.

Table 5. Adaptive permission restoration algorithm.

Input	assigned $PME_{(1...n)} \in$ PMT, $PAE_{(1...m)} \in$ PAT, Current context *C*, PermissionQueue
Output	PAE_z
Procedure	**begin** **while** PME.CR == true set result to 0 read current context *C* **for** *i* = 0 to *n* **do** **if** PME.CR.CRE_i = true **then** result ← result + 1 **end if** **end for** /* CR is true when all CREs are true */ **if** result == *n* **then** PME.CR ← true **else** RDE.CR ← false **end if** **end while** /* restore the permission by updating PAE to PAT utilizing permissionQueue where the previous status are stored */ **if** PME.Condition == disable **then** get PAE_z from PermissionQueue where $PAE_z.p$ == PME.*p* insert PAE_z to PAT **else** get PAE_z from PAT where $PAE_z.p$ == PME.*p* modify $PAE_z.p.A$ to PME.Condition update PAE_z to PAT **end if** **end**

4.3.3 Personalized permission modification

Personalized permission modification modifies an action of a given permission via referencing of the user's preferences in the user profile repository. Thus, for an object performing the same action, changing the object to a user preferred object results in personalized access control. We assumed that the user's preferences are included in the user profile repository. In the user profile repository, the user's preferred objects are determined according to each action, and are located in UPR.*u*.*A*.preferenceObject.

When the action of the permission is the same as the action in the user profile repository, the object of the permission is modified to the user's preferred object via inclusion of the new PAE in PAT. Table 6 is the pseudo-code for personalized permission modification.

Table 6. Personalized permission modification algorithm.

Input	User U, UPR, $PAE_{(1...n)} \in PAT$
Output	PAE_z
Procedure	**begin** **if** $PAE_z.p.A$ == UPR.$u.A$ **then** /* change the preferenceObject in PAE by utilizing the information stored in UPR */ modify $PAE_z.p.O$ to UPR.$u.A$.preferenceObject **end if** **end**

Table 7. Notations for trustworthiness analysis.

Notations	Descriptions
getUpdatedContext()	Fetching current context from CIR
getTableInfo(tableName)	Fetching table information
getQueueInfo(P_i)	Fetching permission P_i from permission queue
getProfileInfo(U_i)	Fetching user profile of user U_i from UPR
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
checkUPR(U_i, P_j)	Check whether action of P_j is existed in UPR of U_i or not

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 = <(Alice, (Location, (ward, String, 302))), positive> CRE_2 = <(John, (Location, (ward, String, 302))), positive> CRE_3 = <(Bob, <(Usage, (room, String, ward)) ∧ (Location, (ward, String, 302))>), positive> CR_{PM} = <CRE_1, CRE_2, CRE_3> PME = <$r_{nurseRole}$, $p_{accessData}$, CR_{PM}, read>

5. TRUSTWORTHINESS ANALYSIS

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

Prior to the analysis, we assume that context requirements for dynamic access control are set to each table as Table 8.

According to Table 8, in RAT, the context requirements for assigning a presenter role to Bob are described when presentation is scheduled to Bob and Bob meets those conditions. In RDT, the context requirements are defined for delegating Bob's roles to John, when Bob goes on a business trip. The context requirements are also specified for revoking roles previously assigned or delegated in PMT. In PMT, the context requirement is specified for modifying the access permission. According to the described context requirement, Alice, a nurse, can access to patient's records only when she locates in the Bob's ward with Dr. John.

Theorem 1 The CA-RBAC scheme guarantees adaptive role assignment, delegation, and revocation according to the change of context.

Proof:

ACM → CIR: getUpdatedContext()
CIR → ACM: updatedContext
ACM → CAUAM: updatedContext
CAUAM: checkCR()
if (CR_{RA} == true) then
  CAUAM → UAT: adaptiveRoleAssign(RAE_i)
  UAT: UAE = <*U*, $RAE_i.r$, none>
else if (CR_{RD} == true) then
  CAUAM → UAT: adaptiveRoleDelegate(RDE_i)
  UAT: $UAE_{(1\ldots i)}$ = <$RDE.u_{delegatee}$, $u_{delegatee}.r_{(1\ldots i)}$, $RDE.u_{delegator}$>
else if (CR_{RA} == false or CR_{RD} == false) then
  CAUAM → UAT: adaptiveRoleRevoke()
  UAT: delete(UAE_i)

Context requirements are checked in CAUAM 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 change of context.

Theorem 2 The CA-RBAC scheme guarantees adaptive modification of permissions and their restoration according to the change of context.

Proof:
ACM → CIR: getUpdatedContext()
CIR → ACM: updatedContext
ACM → CAPAM: updatedContext
CAPAM: checkCR()
if (CR_{PM} == true) then
  CAPAM → PAT: getTableInfo(PME_i)
  PAT → CAPAM: PME_i
  CAPAM → PQ: copy($PME_i.p$)
  CAPAM → PAT: update(PME_i)
  PAT: $PME_i.p$ = <O, PME_i.Condition>
else if (CR_{PM} == false) then
  CAPAM → PAT: getTableInfo(PME_i)
  PAT → CAPAM: PME_i
  CAPAM → PQ: getQueueInfo($PME_i'.p$)
  PQ → CAPAM: $PME_i'.p$
  if (Compare($PME_i.p$, $PME_i'.p$ == false)
    CAPAM → PAT: update(PME_i)
    PAT: $PME_i.p$ = <O, PME'_i.Condition>

Context requirements are checked in CAPAM 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 change of context.

Theorem 3 The CA-RBAC scheme guarantees personalized permission modification based on the individual's preferences stored in the user profile repository.

Proof:
CAPAM → UPR: getProfileInfo(U_i)
UPR → CAPAM: profileInfo
CAPAM → PAT: getTableInfo(PME_i)
PAT → CAPAM: PME_i
if (checkUPR(u_i, $PME_i.p_j$) == true) then
  CAPAM → PAT: update(PME_i, p_j)
  PAT: $PME_i.p_j$ = <UPR.u_i.A_k.O, A_k>

For a permission that performs the same action, the object of the permission must be modified by the user's preferences. This scheme references the user profile repository for personalized access control. If the preferred object for the same action is in the user profile repository, the current object is replaced by the preferred object in the user profile,

enabling personalized access control. Thus, this scheme guarantees personalized permission modification based on the individual's preferences stored in the user profile repository.

6. CONCLUSION

In this paper, we proposed a context aware RBAC (CA-RBAC) scheme in ubiquitous computing environments. In our scheme, we provided context aware UA and PA based on the satisfaction of context requirements. Context requirements were defined in each table, enabling us to construct a more detailed description. We also provided various access control algorithms including role assignment, role delegation, role revocation, permission modification, and permission restoration. We also guaranteed personalized access control that considers the user's preferences.

REFERENCES

1. X. Feng, X. Jum, H. Hao, and X. Li, "Context-aware role-based access control model for web services," in *Proceedings of International Workshops on Information Security and Survivability for Grid*, LNCS 3252, 2004, pp. 430-436.
2. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control*, Artech House Publishers, Boston, 2003.
3. D. Kulkarni and A. Tripathi, "Context-aware role-based access control in pervasive computing systems," in *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, 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," *Lecture Notes in Computer Science*, Vol. 3528, 2005, pp. 236-242.
5. G. Zhang, "Dynamic context aware access control for grid applications," Master Thesis, Department of Electrical and Computer Engineering, 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 *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, 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," in *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, Vol. 4443, 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," in *Proceedings of ACM Symposium on Access Control Models and Technologies*, 2001, pp. 171-181.
9. X. Tang, Y. Zhang, and J. You, "RCACM: Role-based context coordination model for mobile agent applications," in *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing*, 2003, pp. 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," in *Proceedings of ACM Symposium on Access Control Models and Technologies*, 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," in *Proceedings of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004, pp. 159-163.
12. H. Feinstein, R. Sandhu, E. Coyne, and C. Youman, "Role-based access control models," *IEEE Computer*, Vol. 29, 1996, pp. 38-47.

Jung Hwan Choi received the B.E. and M.E. degrees from Sungkyunkwan University in 2007 and 2009, respectively. He is currently working at Samsung Electronics Co., Ltd., as a software engineer. His research interests include pervasive computing, ubiquitous middleware, and mobile agent systems.

Hyunsu Jang received the B.S. and M.S. degrees from Sungkyunkwan University in 2002 and 2005, respectively. He is currently studying for the Ph.D. degree at the Department of Electrical and Computer Engineering, Sungkyunkwan University. His research interests include mobile agent systems, HCI computing, and ubiquitous middleware.

Young Ik Eom received his B.S., M.S. and Ph.D. degrees from the Department of Computer Science and Statistics of Seoul National University in Korea, in 1983, 1985, and 1991, respectively. From 1986 to 1993, he was an Associate Professor at Dankook University in Korea. He was also a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine from Sep. 2000 to Aug. 2001. Since 1993, he is a Professor at Sungkyunkwan University in Korea. His research interests include system software, distributed computing, mobile agent systems, and system securities.