A Context-Aware Access Control Model for Pervasive Computing Environments

Sareh Sadat Emami
Electrical Engineering Department, Khaje Nasiredin Tusi University of Technology, Tehran, Iran
emami@ee.kntu.ac.ir

Morteza Amini
Network Security Center, Sharif University of Technology, Tehran, Iran
m_amini@ce.sharif.edu

Saadan Zokaei
Electrical Engineering Department, Khaje Nasiredin Tusi University of Technology, Tehran, Iran
szokaei@eetd.kntu.ac.ir

Abstract

In pervasive computing environments, a user can access resources and services from any where and at any time; thus a key security challenge in these environments is the design of an effective access control model which is aware of context modifications. Changes in context may trigger changes in authorizations. In this paper, we propose a new context-aware access control model based on role-based access control model for pervasive computing environments. We assign roles to users dynamically based on the long-term context information and tune active role's permissions according to the short-term context information of the users and environment.

1. Introduction

Vision of Ubiquitous Computing was described by Mark Weiser in 1991 [1]. He said, "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". In pervasive computing environments, users may access resources and services remotely. There are different sorts of users and services and all of them are not predefined [2]. Furthermore, context plays a crucial role in these environments and affects decision making processes significantly.

Because of the users being mobile and numerous, context is a considerable factor in access control. Dey and Abowd [5] defined context as any information that can be used to characterize the situation of an entity. An entity would be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves. Context may include date, time, location, system capabilities and other information about entities and environment. Context information may be variable over time, thereby traditional access control models cannot comply all the requirements in these environments [4]. Context sensitive authorizations are more applicable in pervasive computing environments than traditional access controls, because:

- They are very flexible through using context information for access control.
- Different security levels are possible for an access in these systems. Users' intents and behaviors also affect their access to services in addition to context information.

Role-Based Access Control (RBAC) [6] model is proposed by NIST in 1996. The definitions are explicit; hence implementation of this model is trivial. The comprehensive framework for RBAC models is characterized as follows:

- RBAC0: the basic model in which users are associated with roles (U-A) and roles with permissions (P-A).
- RBAC1: RBAC0 with role hierarchy.
- RBAC2: RBAC0 with constraints on role and permission assignments.
- RBAC3: combination of RBAC1 and RBAC2.

Figure1. shows the relational diagram of RBAC3[1].

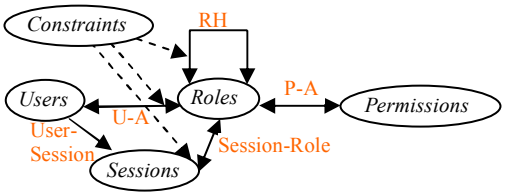

Figure 1. RBAC relational diagram

In our approach, the RBAC model is modified in order to be compatible with these environments. As described in section 2, all access control approaches in pervasive computing use context for controlling authorization. Some of them use rules for access control; but considering lots of objects and subjects in pervasive computing environments, rule based access control is not an effective solution. Most of proposed models for such environments are based on RBAC;

[1] From now on RBAC

0-7695-3006-0/07 $25.00 © 2007 IEEE
DOI 10.1109/IPC.2007.28

they limit role-permission assignment using context, but they do not use context for static role assignment to users. So in these models, there are lots of roles being hard or almost impossible to manage.

We define prerequisite context for role assignments and enable dynamic role-assignment in this model. In the beginning of a session, roles are assigned to a user according to the context information. For making dynamic authorization possible, active role's permissions are overridden for each user in his/her session by changing the context information. For assigning permissions to a role, we need some context information about the user and the environment as a precondition and if all preconditions are confirmed, these permissions are granted to the user playing that role.

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 context-aware access control model is presented in section 3. Section 4 discusses evaluation of the model and section 5 concludes the paper while stating some future works.

2. Related Works

Context sensitive access control based on RBAC is proposed in [4]. The proposed model has predefined roles for context management. There are four roles defined in their framework: Context Owner (CO), Context Provider (CP), Context Broker (CB) and Context-Aware Service Provider (CASP). They focused on context information assurance and secure transmission of context between the pervasive nodes.

Zhang and Parashar [8] proposed a dynamic RBAC model that extends the RBAC and dynamically adjusts static role and permission assignments based on context information. Central authorizer assigns a role state machine to each user's agent and changes the active role in the state machine according to the changes in user context. Each object has a permission state machine that is modified when the context changes for system roles.

Cerberus, a context-aware security scheme for smart spaces, is proposed in [3]. 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 [9]) 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.

3. Context-Aware Access Control Model

In this section, we present our proposed model, named CAP for controlling accesses to resources in pervasive computing environments. In CAP, context predicate is represented as a 4-tuple <entity, context type, context relater, value> following the Gaia project's proposition [9]. This representation determines the value of an entity's context according to the relater. Entity is a user or an environmental entity that its context is important for authorization. For example, context information predicate <Bob, Location, "=", library> with triple context <Location, "=", library> describes that Bob is in library or <env,Temperature, "=", 23> explains that the environment temperature is 23 degrees of Celsius. We can present the entity context set as a formal expression as is presented in (1). Some context types appertain to users such as location and finger print while some other ones appertain to environment, such as temperature and time. In (1), CtxValSet is a set of context values. For each context type, its possible values are a subset of allowable context values.

$$EntCtxSet = EntSet \times CtxSet \quad (1).$$

$EntSet = Users \cup EnvEntity$

$Users$ = *set of users*

$EnvEntity = \{env\}$

$CtxSet \subseteq CtxTypeSet \times CtxRelaterSet \times CtxValSet$

$CtxTypeSet$ = *set of context types*

$CtxRelaterSet = \{"=", "\neq ", ">", "<", "\leq ", "\geq "\}$

$CtxValSet$ = *possible values of all context types*

We divide context information into Long-Term context (LTC) and Short-Term context (STC), as follows:

$$CtxSet = LTC\text{-}Set \cup STC\text{-}Set \quad (2).$$

$$LTC\text{-}Set = E\text{-}LTC\text{-}Set \cup U\text{-}LTC\text{-}Set$$

$$STC\text{-}Set = E\text{-}STC\text{-}Set \cup U\text{-}STC\text{-}Set$$

Long-Term Context (LTC) is the one that its value does not change in a time period, named μ times of average session lifetime, such as age, weight and system capabilities. The μ value must be selected carefully to ensure that the probability of changing the Long-Term context information during a session is trivial. Selection of LTCs depends on the environment and session lifetime. Assume average session duration is less than 1 or 2 hours and μ is 3, so we can select date as a LTC. LTC-Set contains two sets of LTCs: Environmental LTC (E-LTC-Set) and User LTC (U-LTC-Set). Another set contains Short-Term Contexts (STC) that maybe changed during a session, such as time, location and CPU load. STC-Set also includes

Environmental STCs (E-STC-Set) and User STCs (U-STC-Set).

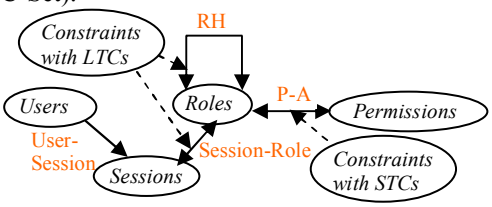

Figure 2. CAP relational diagram

We apply constraints to model in two levels. In the first level, we apply constraints to role-hierarchy and session-role assignment with LTCs. In the second level, the role-permission assignment is limited with STCs. CAP relational model is depicted in figure 2.

3.1. Model Definition

We categorize current context information into the following groups: LTCI as in (3), which is current Long-Term Context Information, and STCI as in (4), that is current Short-Term Context Information.

$LTCI \in P(EntSet \times LTC\text{-}Set)$ (3).

$STCI \in P(EntSet \times STC\text{-}Set)$ (4).

We have RBAC formal definitions [6] as U: users set, R: roles set, Prm: permissions set and S: sessions. Permission is a 2-tuple of object and access right such as <book, read>.

CAP assigns roles to users according to LTCI; so assigning each role needs prerequisite LTCs. Role Assignment Condition (RAC) maps a subset of Environmental LTC-Set and User LTC-Set to each role as in (5):

$RAC: R \rightarrow P(U\text{-}LTC\text{-}Set) \times P(E\text{-}LTC\text{-}Set)$ (5).

Henceforth, for every $(r, (USet, ESet)) \in RAC$, we use $RAC(r).U\text{-}LTC\text{-}Set$ to refer to $USet$ and $RAC(r).E\text{-}LTC\text{-}Set$ to refer to the $ESet$.

Each session belongs to a user that has some roles, and the S-U is a mapping function that assigns a user to a session as defined in (6):

$S\text{-}U: S \rightarrow U$ (6).

For assigning a role to a user session, CAP checks prerequisite LTCs of this role. In (7), Session-Role (S-R) is mapping function defined for this aim.

$S\text{-}R: S \rightarrow P(R)$ (7).

$S\text{-}R(s_j) = \{rl \in R \mid \forall <t,r,v> \in RAC(rl).U\text{-}LTC\text{-}Set,$
$[<S\text{-}U(s_j),t,r,v> \in LTCI] \wedge \forall <t',r',v'> \in RAC(rl).E\text{-}LTC\text{-}Set , [<env,t',r',v'> \in LTCI]\}$

Thus we assign roles to a user dynamically when his/her session starts. For assigning permission to a role, some STCs must be checked as preconditions. In CAP, we have prerequisite conditions as STCs for role-permission assignment that are defined statically in the core of the model. It means CAP overrides role-permission assignment according to the STCs defined before. Role-Permission Condition (RPC) is a mapping function that assigns a subset of E-STC-Set and U-STC-Set to each role with a specific permission as in (8).

$RPC: R \times Prm \rightarrow P(U\text{-}STC\text{-}Set) \times P(E\text{-}STC\text{-}Set)$ (8).

Henceforth, for every $(r,p, (USet, ESet)) \in RPC$, we use $RPC(r,p).U\text{-}STC\text{-}Set$ to refer to $USet$ and $RPC(r,p).E\text{-}STC\text{-}Set$ to refer to $ESet$.

For each user, when the session starts and after assigning roles to that session, CAP obtains permissions of the session roles and their preconditions for the session. Session-Permissions Assignment as defined in (9) is a mapping function from session to the permissions of user's roles in the session and their prerequisite STCs.

$SPA: S \rightarrow P(Prm \times P(U\text{-}STC\text{-}Set) \times P(E\text{-}STC\text{-}Set))$ (9).

$SPA(s_j) = \{(p, U\text{-}Set, E\text{-}Set) \mid \exists rl \in S\text{-}R(s_j) \wedge \exists p \in Prm, [U\text{-}Set = RPC(rl,p).U\text{-}STC\text{-}Set \wedge E\text{-}Set = RPC(rl,p).E\text{-}STC\text{-}Set]\}$

From now for every (s, (p, (USet, ESet))) $\in$ SPA, we use SPA(s).P-Set to refer to p, SPA(s)(p).U-STC-Set to refer to USet and SPA(s)(p).E-STC-Set to refer to ESet.

For every user's access request in a session, CAP checks conditions for requested permission, if all of them are satisfied, permits user's access. In (10) Request-Authorization is a mapping function that assigns "Grant" or "Deny" as a response to the session and requested permission, according to permission conditions and current Short-Term contexts (STCI).

$Rq\text{-}Au: S \times Prm \rightarrow \{Grant, Deny\}$ (10).

$$Rq\text{-}Au(s,p) = \begin{cases} Grant, & if\ p \in SPA(s).PSet \wedge \\ & \forall <t,r,v> \in SPA(s)(p).U\text{-}STC\text{-}Set, \\ & [<S\text{-}U(s),t,r,v> \in STCI] \wedge \\ & \forall <t',r',v'> \in SPA(s)(p).E\text{-}STC\text{-}Set \\ & , [<env,t',r',v'> \in STCI] \\ Deny, & otherwise \end{cases}$$

3.2. Architecture

There are two main parts in our proposed architecture for the CAP access control model: Domain Authority (DA) and Session Agent (SA). In figure 3, the architecture is shown.

There is a DA in each system domain and when a user enters the domain and starts a session, DA sets up a SA for that user.

Domain Authority aggregates Long-Term contexts and assigns roles to a user in the beginning of a session depending on LTCs and prerequisite conditions for that role, so DA appoints S-R in a session. According to user's session roles and RPCs, DA appoints SPA and gives it to SA for controlling accesses.

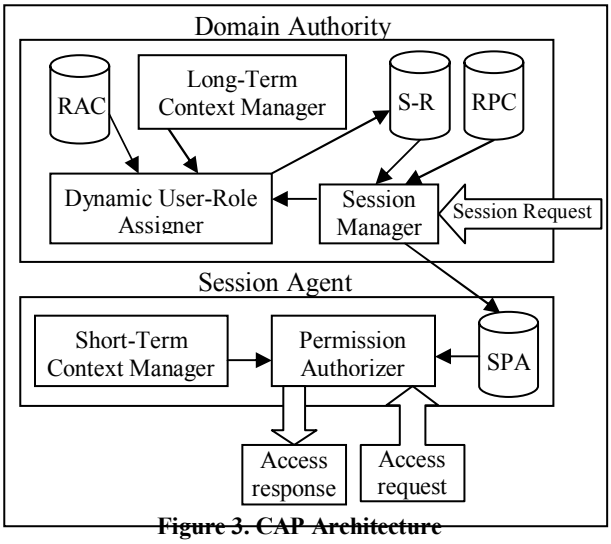

Figure 3. CAP Architecture

There are static databases in the model including: Role-Permission Conditions and Role Assignment Conditions and there is a dynamic database as Session-Role, that maintains all users' sessions roles.

Now, we explain the basic components of the Central Authority:

- **Long-Term Context Manager:** Aggregates LTCI from sensors of environment and users, interprets them and stores them in a specific format.
- **Session Manager:** Receives session request from users, assigns a session and a SA to a user and asks Dynamic User-Role Assigner to determine user's roles in this session and according to assigned roles fills SPA for the Session Agent.
- **Dynamic User-Role Assigner:** Assigns roles to a user's session according to LTCI and RAC and fills S-R database.

Session Agent: Collects Short-Term contexts and evaluates each user's access request according to SPA. If the requested permission is accepted by Request-Authorization function, the access is permitted, otherwise it is denied.

There is a dynamic database as SPA that DA fills it. Main components of this part are as follows:

- **Short-Term Context Manager:** Works the same as Long-Term Context Manager Component in DA, but it collects STCI.
- **Permission Authorizer:** Makes a decision about user's access request according to its roles permissions in the session and the required context information for these permissions in SPA database.

3.3. Case Study

In this section, a simple example is demonstrated for making CAP model clearer. Although this example is small, it provides enough insight into the process and assists comprehending the model.

Figure 4 shows an online examination scenario, depicted as an access sequence chart.

In this scenario, there are two roles: teacher and student, and one object: exam documents. We assume that Bob is a teacher and Alice is a student.

You can see the definition of prerequisite LTCs for roles in figure 5; f1 is a teacher's valid finger print, "192.167.16.3" is a registered IP address for an online student and "8423641" is a valid student ID.

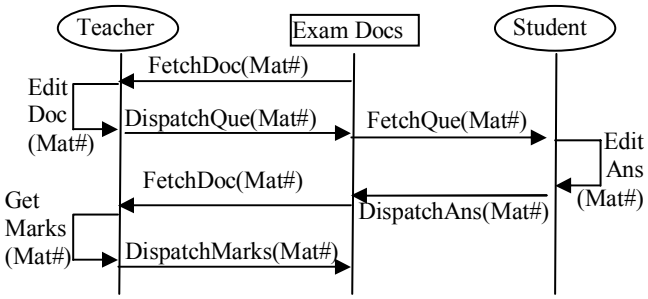

Figure 4. A sample scenario

We have permission set in figure 6 and RPCs in figure 7. RPC is a null set if the permission of the role does not have prerequisite contexts.

RAC (teacher) = {<Finger-Print,"=", f_1>}, {}
RAC (student) = {<IP-Address,"=", 192.167.16.3>,
<Student-ID,"=", 8423641>}, {}

Figure 5. Role-Assignments Conditions

Prm = { <ExamDoc, Fetch>, // as P-F
<ExamDoc, EditQuestions>, // as P-EQ
<ExamDoc, DispatchQuestions>, // as P-DQ
<ExamDoc, EditAnswers>, // as P-EA
<ExamDoc, DispatchAnswers>, // as P-DA
<ExamDoc, GetMarks>, // as P-GM
<ExamDoc, DispatchMarks>} // as P-DM

Figure 6. Permission Set

Bob must design questions for the exam before the exam date.

At the first session, Bob, using his client PC, fetches the exam document with his matriculation number, edits and then dispatches it to the exam document server. At the second session, Alice fetches the exam document by her matriculation number, answers to the questions by editing the document and then dispatches it to the exam document server. The third session has occurred after the exam, when Bob fetches the documents, evaluates the answers and then uploads the grades to the exam server.

RPC (teacher,P-F) = {<Mat#, "=", ExamDoc#>}, {}
RPC (teacher,P-EQ) = { <Mat#, "=", ExamDoc#> },
{<Date, "<", ExamDate>}
RPC (teacher,P-DQ) = { <Mat#, "=", ExamDoc#>},
{<Date, "<", ExamDate>}
RPC (teacher,P-GM) = {<Mat#, "=", ExamDoc#>},
{<Date, ">",ExamDate>,<Date,"<", MarksDeclaringDate>}

RPC (teacher,P-DM) = { <Mat#, "=", ExamDoc#>},
{<Date,">",ExamDate>, <Date,"<",MarksDeclaringDate >}

RPC (student,P-F) = {<Mat#, "=", ExamDoc#>},
{<Date, "=", ExamDate>, <Time, "≥ ", StartExamTime>,
<Time, "≤ ", EndExamTime>}
RPC (student,P-EA) = {<Mat#, "=", ExamDoc#> ,
<Location, "=", ExamRoom>}, {<Date, "=", ExamDate>,
<Time, "≥ ", StartExamTime>, Time,"≤ ",EndExamTime>}

RPC (student,P-DA) = {<Mat#, "=", ExamDoc#>,
<Location, "=", ExamRoom>},
{<Date, "=", ExamDate>, <Time, "≥ ", EndExamTime>,
<Time, "≤ ", SubmissionDeadline>}

Figure 7. Role-Permission Conditions

At start, we have LTCI as in figure 8; we assume it is the same in all three sessions.

LTCI = {<Bob, Finger-Print, "f_1">,
<Alice, Student-ID, "=", 8423641>,
<Alice, IP-Address, "=", 192.167.16.3>}

Figure 8. Long-Term Context Information (LTCI) for all sessions

At the first session, Bob wants to design exam questions. He begins the session and according to the LTCI, *S-R(s_1) = {teacher}*, he gets the teacher role. According to RPCs and Bob's role, CAP assigns permissions to his session, which is shown in figure 9. When he wants to access the Exam Document, CAP fetches permission *P-F* and its STC conditions from *SPA(s_1)*; so if Bob's matriculation number is ExamDoc number and *Rq-Au(s_1,P-F)= "Grant"* he can fetch ExamDoc. Also, if the date is before the exam date, he can edit the questions and dispatch ExamDoc. It means *Rq-Au(s_1, P-EQ) ="Grant"* in second request and *Rq-Au (s_1, P-DQ) = "Grant"* in third request.

At the second session, Alice wants to take an exam. She gets the student role, i.e., *S-R (s_2) = {student}*, according to LTCI. Figure 10 shows *SPA (s_2)* for this session. If her matriculation number is ExamDoc number, today is the exam date and time of access request is in exam duration then *Rq-Au (s_2, P-F) ="Grant";* it means She can fetch ExamDoc. Also if her location is the exam room, she can answer to the questions, because *Rq-Au (s_2, P-EA) ="Grant"*.

She can dispatch answers if her matriculation number is ExamDoc number, date equals exam date, her location is exam room and time is before the submission deadline and after the end of the exam duration; So the *Rq-Au (s_2, P-DA) ="Grant"* is approved.

SPA (s_1) = SPA (s_3) =
{
(P-F, {<Mat#, "=", ExamDoc#>}, {}),
(P-EQ, {<Mat#, "=", ExamDoc#>},
{<Date, "<", ExamDate>}),
(P-DQ, {<Mat#, "=", ExamDoc#>},
{<Date, "<", ExamDate>}),
(P-GM, {<Mat#, "=", ExamDoc#>},
{<Date, ">", ExamDate>,
<Date, "<",MarksDeclaringDate >}),
(P-DM, {<Mat#, "=", ExamDoc#>},
{<Date, ">", ExamDate>,
<Date, "<",MarksDeclaringDate >})
}

Figure 9. Session-Permission Assignment for first session and third session

SPA (s_2) =
{
(P-F, {<Mat#, "=", ExamDoc#>},
{<Date, "=", ExamDate>,
<Time, "≥ ", StartExamTime>,
<Time, "≤ ", EndExamTime>}),
(P-EA, {<Mat#, "=", ExamDoc#>,
<Location, "=", ExamRoom>},
{<Date, "=", ExamDate>,
<Time, "≥ ", StartExamTime>,
<Time, "≤ ", EndExamTime>}),
(P-DA, {<Mat#, "=", ExamDoc#>,
<Location, "=", ExamRoom>},
{<Date, "=", ExamDate>,
<Time, "≥ ", EndExamTime >,
<Time, "≤ ", SubmissionDeadline>})
}

Figure 10. Session-Permission Assignment for second session

At the third session, Bob wants to evaluate the exams. In this session, he gets the teacher role again; thus *S-R (s_3) = {teacher}* and *SPA (s_3)* is acquired same as first session (figure 9). In first request he wants to fetch the ExamDoc, so If his matriculation number is ExamDoc number, *Rq-Au (s_3,P-F)="Grant"* and he accesses to ExamDoc. Also in second and third requests if today is after the exam date and before the marks declaring date, *Rq-Au (s_3,P-GM)="Grant"* and *Rq-Au(s_3,P-DM)="Grant"*, thus he can evaluate the exams and dispatch the marks.

4. Evaluation

RBAC is a static model; it defines user-permission assignment and role-permission assignment statically. Some extensions of RBAC such as DRBAC [8] tried to create a dynamic model based on RBAC by adding context awareness, but most of them use context as conditions for role-permission assignment and they don't have dynamic user-role assignment. CAP not

only controls accesses to the objects and assigns permissions to the roles according to the context information, but also assigns roles to users dynamically depending on their context in each session.

For better evaluation, we collected some proposed factors from other publications and evaluated our model using them.

- Reloading context is time consuming and inefficient, so an effective model must decrease the response time. By dividing context to Long-Term and Short-Term context, we can improve time complexity of operations, because LTCs are checked at the beginning of a session and we do not need to check it during a session. Also CAP just checks the prerequisite STCs for requested permissions in a session. Therefore, the average response time of access requests have been decreased in this model.
- In pervasive computing environments, security service itself has to be ubiquitous [3]. CAP can be ubiquitous by its architecture such as proposed architecture. CAP by distributed DAs and SAs can control access to services at any time and from any where; so it has ubiquitous security services.
- Access control model must be scalable [7]. We can use CAP in large scale networks; its roles are variable and we can add new roles to the model with a little cost; also users are not fix in the model. These flexibilities help CAP to be scalable. In addition, CAP is used in distributed networks, where we have a DA for each domain that interferes at the start of the sessions and during the sessions, SAs control accesses.

5. Conclusion and Future works

In this paper, we proposed a context aware access control model based on RBAC. This model can assign roles dynamically to users and limit their access with context information. We described our model in a formal manner and presented a simple case study to demonstrate the applicability of the model. An architecture was proposed according to the model to help the implementation of an access control system based on the CAP model.

We used context information for controlling accesses but did not discuss about their management. For managing context information, we need a secure context infrastructure. Context information must be updated quickly. Since the integrity of information must be assured, quality of context information is very important. An attacker must not be able to forge the context information.

Intents and goals of the users are related to their behaviors. Knowledge of past accesses may allow us to infer present (or future) behavior of users. Therefore, with aggregating the history of a user's behaviors, we can have an effective access control.

Another future work is developing a mechanism to implement our model more realistically in large-scale applications and evaluating the performance and scalability of the designed context-aware access control system.

6. Acknowledgment

We wish to thank Dr. Rasool Jalili for his contribution. This work was partially supported by Iran Telecommunication Research Center.

7. References

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," IEEE Personal Communication, Aug. 2001, pp. 10-17.

[2] T. Kagal, L. Finin, and A. Josh. "Trust-based security in pervasive computing environments," IEEE Computer Society Press, December 2001, pp. 154-157.

[3] Jalal Al-Muhtadi, Anand Ranganathan, Roy Campbell and M. Dennis Mickunas, "Cerberus: A Context-Aware Security Scheme for Smart Spaces," in Proceedings of the First IEEE Annual Conference on Pervasive Computing and Communications (PerCom 2003), Fort Worth, Texas, March 2003, pp. 489-496.

[4] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. Ebben, and J. Reitsma, "Context sensitive access control," In Proceedings of the 10th ACM SACMAT, Sweden, June 2005, pp. 111-119.

[5] A. K. Dey and G. D. Abowd, "Towards a better understanding of context and context-awareness," Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, June 1999.

[6] R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, "Role-based access control models," IEEE Computer Society Press 29(2), pp. 38–47, 1996.

[7] Kazuhiro Minami and David Kotz, "Secure Context-sensitive Authorization," Journal of Pervasive and Mobile Computing (PMC), March, 2005, pp. 123-156.

[8] 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 (CNDS 2004), San Diego CA, USA, January 2004, pp. 219-225.

[9] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," IEEE Pervasive Computing, Piscataway, NJ, USA, October 2002, pp. 74-83.