International Journal of Computer Theory and Engineering, Vol. 2, No. 1 February, 2010
1793-8201

Validation of Temporary Delegation and Revocation of Roles with UML and OCL

Mohsin Ali Memon, Manzoor Hashmani and Karsten Sohr

***Abstract*—Organizations having a large number of employees face several difficulties to separate job assignments to individual users. The situation becomes more complex when job assignments are delegated to those users who fulfill some explicit conditions. The delegation models developed so far discuss various issues regarding delegation of roles, but no mechanism has been developed to specify and validate constraints which are applied in course of temporary delegating (temporal delegation) and revoking a role to and from a user. This paper proposes a validation mechanism for flexible delegation and revocation of job roles to and from users with specific conditions. Also, we attempt to specify and validate n-level delegation and cascading/non-cascading revocation processes in an organization.**

I. Introduction

Security has been a major issue in the development of software for networked organizations. As the number of employees of an **organization** may vary with variable job assignments, therefore it becomes complicated to manage job assignments to various users. To cope with such situations Role Based Access Control Model (RBAC) [1] was proposed for advanced access control as it reduces complexity and cost of security administration in specifically those applications which are networked and are accessed by large number of users with variable job assignments. In various organizations, users have permissions to delegate their job rights to other users at specific time periods with some restrictions. In the same way users can revoke the delegated rights after some time period.

Almost all major organizations facilitate their users to delegate their job assignments to other users when work load of users cross certain limits or when the authorized user is unavailable. Therefore, delegation of rights and roles are followed by certain constraints which must be accomplished and only after that, job assignments be delegated to other users. For example, Ahmed delegates task "review voucher" to Mariam on wednesdays during a staff meeting or Dr. Sohail delegates "treat patient" task to a nurse when he is at home.

The delegation models developed by [3], [4], [5] deal with on delegation of rights to other users, but have not specified any constraint in a formal language which limit temporal delegation of rights of one user to others. Therefore, it is required to develop a mechanism to specify and validate constraints applied on temporary delegation of roles and revocation of roles, that is, for short period of time when the actual user is engaged in some other activity or not available, with the help of a formal language so as to ensure that job assignments are granted and revoked accurately.

In this article, we propose a delegation/revocation mechanism with the help of pre conditions and post conditions applied on delegation and revocation methods of an organization. In addition, the reason behind this paper is to specify constraints on delegation in a formal language and validate them with the help of a tool. We use the Unified Modeling language (UML) and the Object Constraint Language (OCL) for the specification of delegation and revocation schemes. We validate the delegation constraints with the help of the USE tool (UML-based specification environment), a validation tool for UML models and OCL constraints [2].

This paper is organized as follows. In Section 2, we briefly describe related technologies. Section 3 discusses the Specification and Validation of Delegation and Revocation of Authority in USE. Several related works are discussed in Section 4. Section 5 emphasizes on future work and concludes the paper.

II. Related Technologies

A. Role-Based Access Control

Role-Based Access Control (RBAC) is a powerful access control model introduced by Ferraiolo and Kuhn [13]. Furthermore, an ANSI standard proposed by the proposed by the National Institute of Standards and Technologies (NIST) in 2001 exists [1]. RBAC is one of the most pervasive security models used these days which has replaced traditional discretionary (DAC) and mandatory (MAC) access control. It helps to reduce the complexity of access control administration, specifically when dealing with large systems. The RBAC model has four main components – users, roles, permissions and sessions. A role is generally representation of organizational functions. Users are assigned to roles and permissions are assigned to roles as well. Users are assigned all their permissions according to their role memberships. Users interact with the system through sessions and roles are assigned to a particular sessions as well.

In RBAC, roles represent functions within a given

Mohsin Ali Memon* is with is with the Dept. of Computer Systems and Software Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan, email: memohsin@gmail.com

Manzoor Hashmani** is with is with the Dept. of Computer Systems and Software Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan, email: mhashmani@yahoo.com

Karsten Sohr*** is with Dept. of Mathematics & Computer Science, University of Bremen, Germany, e-mail: sohr@tzi.de).

organization; authorizations/permissions are granted to roles instead of to single users. Authorization constraints are an important aspect of RBAC and are sometimes argued to be the principal motivation for RBAC. The common examples of RBAC authorization constraints include static separation of duty, dynamic separation of duty, prerequisite roles, and cardinality constraints.

B. *Object Constraint Language*

OCL is a formal language used to describe expressions on object-oriented models. These expressions typically define invariant or pre- and postconditions that must hold for the system being modeled or queries over objects described in a model. OCL is used by UML modelers to specify application specific constraints in their models. It is a text-based language which makes it easier to comprehend and specify constraints on UML models and thus reduces the ambiguity while retaining the understandability of models. The use of OCL is not limited to UML only as it can also be used by other languages, notations, methods and software tools in order to specify constraints and other expressions of their models. There are different tools which support the specification of UML models and the validation of OCL constraints applied on these models [6]. In an OCL expression, the reserved word `self` is used to refer to a contextual instance. The type of the context instance of an OCL expression is written with the context keyword, followed by the name of the type. The label `inv:` declares the constraint to be an invariant. Invariants are conditions that must be true during the lifetime of a system for all instances of a given type.

C. *USE Tool*

The USE tool [7] allows software modelers to specifiy UML models and validates OCL constraints by checking snapshots of a system. USE is the only OCL tool which allows interactive monitoring of OCL invariants and pre- and postconditions and the automatic generation of non-trivial system states [8]. It is available as Open Source and provides the facility of executing state manipulation commands with the help of the USE API's `executeCmd`() method available in the `MSystem` class. The USE tool facilitates developers in analyzing the model structure and model behavior by generation of typical snapshots and by execution of typical operation sequences. The constraints specified in the UML model are formally checked by developers against their expectations and as a result, they can derive formal model properties.

The USE tool can be employed in various ways in the context of RBAC policies. Specifically, it can be used for the specification and for the validation of RBAC policies in the design phase [9]. In the USE tool we specify our UML model and OCL constraints using a textual description. When we open that text file in USE, it checks the description and verifies a specification against the grammar of the specification language, basically a superset of OCL extended with language constructs for defining the structure of the model. After all these checks are passed, the model can be displayed by the graphical user interface provided by the USE system. USE also offers a project browser which displays all the classes, associations, invariants, and pre- and post-conditions of the current model.

The term delegation refers to the assignment of a particular right or role to another user and revocation is the process by which a delegation that was accepted is removed or retracted [10], [11]. Delegation is a decentralized approach in modern distributed and networked systems to authorize another entity to access the resources in contrast to the traditional centralized mechanism where a security officer was assigned the job to manage sharing of resources and information. In large role-based systems, the users may be in tens or hundreds of thousands and the number of roles may be in the hundreds or thousands. In addition, today's dynamic and collaborative work environment may require users assuming temporary roles.

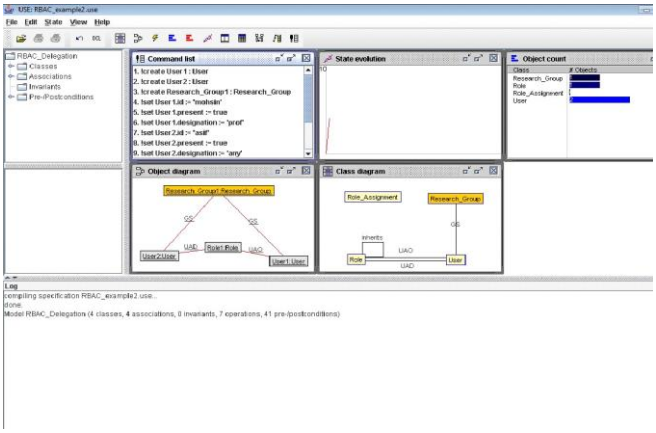

Figure1 The USE Tool

Therefore, managing user assignments is a cumbersome task and could not practically be centralized to a small group of security officers. It is necessary to decentralize the administration through a delegation mechanism in order to increase the scalability of role-based systems. The basic idea behind a role-based delegation is that users themselves may delegate role authorities to other users to carry out some functions authorized to the former.

D. *Delegation and Revocation Model*

This section provides a brief introduction of the delegation and revocation model proposed by Nguyen et al. [4]. According to them, a user can only delegate his role to other user if the following four conditions are satisfied;

a The user (delegator) has the right to delegate his role
b The delegatee must satisfy certain restrictions before the role is delegated to him. In our scenario discussed in Section 3, the delegatee must belong to same research group as of the delegator.
c The delegation depth of the delegatee must be less than the delegation depth of the delegator.
d There should be a maximum validity period of delegation made by the delegator.

In the same manner if a user needs to revoke a role from other users, he can use cascading and non-cascading

revocation. In non-cascading revocation, the following four conditions must be satisfied;

a The user (revoker) has the *right to revoke* the role.
b The revocation must satisfy certain restrictions before the role is revoked from the user who owns it.For example, the delegator must be a professor to revoke the role from the delegatee.
c The revocation must not affect any existing delegation that is dependent on the revoked delegation.
d There should be some time period defined after which the revocation takes effect.

For cascading revocation, the following four conditions must be satisfied,

a The user (revoker) has the right to revoke the role.The revocation must satisfy certain restrictions before the role is revoked from the user who owns it. For example, the delegator must be a professor to revoke the role from the delegatee.
b The revocation must remove all existing delegations that are dependent on the revoked delegation.
c There should be some time period defined after which the revocation takes effect.

The role-based delegation and revocation model proposed by Zhang et al. [11], discusses two types of users,

- *Original users* are those users who are assigned to the role *r:*
- *Delegated users* are those users who are delegated to the role *r:*

This results in two types of user assignment, one is original user assignment (UAO) which is a many-to-many user assignment relation between original users and roles and the second is delegated user assignment (UAD) which is a many-to-many user assignment relation between delegated users and roles.

III. Specification And Validation Of Delegation And Revocation Of Authority In Use

This section discusses our approach of specifying a model example in OCL. Here we specify the delegation and revocation schemes discussed in section 2.4 formally with OCL and then with USE tool we validate our temporal delegation/revocation mechanism with the USE tool. Here we do not use the graphical user interface of USE, but in fact the USE API for several reasons. First, we need to check the pre- and post conditions of the delegation/revocation methods called by a delegator. Secondly, we have to add some dynamic invariants in the model to ensure that the roles are temporarily delegated.

A. Specification of Delegation and Revocation of Authority with OCL

In the following, we describe our formal approach for the specification of delegation and revocation schemes with OCL. Here we take an example scenario of a Research Group containing several research members. The class diagram of this example is shown in Fig. 2. This example will be used in the rest of the paper. We specify our model in USE containing User, Role, Role_Assignment and Research_Group as four major classes used for explanation of our delegation/revocation mechanism. Here, the User class is used for creating objects of all research members and objects of Role class contain sets of permissions to be assigned to research members. The User class has two delegation methods delegaterole_O() and delegaterole_D() for original role assignment and delegated role assignment respectively. Each research member must belong to a research group and information regarding role assignment (both UAO and UAD) to all members along with delegation level is retained by objects of the Role_Assignment class. It is utilized for multilevel delegation which contains all users with their roles assignments along with the delegation level for each role.

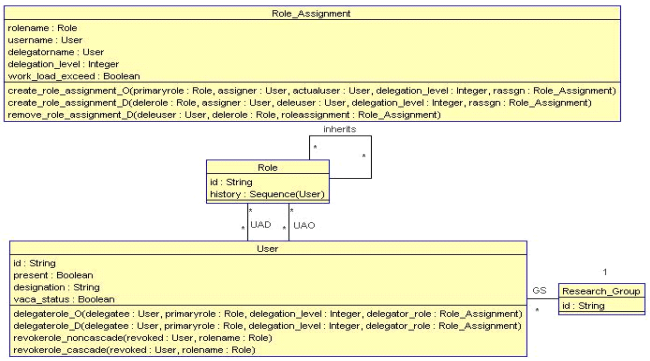

Figure2 Class Diagram of the current Scenario

We specify the constraints for the delegation and revocation model discussed in section 2.4 in OCL as pre- and postconditions of a method named `delegaterole_D()` in Fig.3.

```
User::delegaterole_D(delegatee:User,primaryrole:Role,delegation_level:Integer,
delegator_role:Role_Assignment)

pre delerolepre1: self.role_O->includes(primaryrole)
pre delerolepre2: delegator_role.delegation_level >0          } Delegation Condition 'a'
pre delerolepre3: self.designation='prof'

pre delerolepre4: delegatee.role_D->excludes(primaryrole)

pre delerolepre5: self.group=delegatee.group                  Delegation Condition 'b'

pre delerolepre6: delegation_level<delegator_role.delegation_level   Delegation Condition 'c'


post delerolepost1: delegatee.role_D->includes(primaryrole)
post delerolepost3:
delegator_role.delegation_level=delegator_role.delegation_level@pre-1
```

Figure3 Pre-Postconditions of Delegation Method

Here the user with the UAO association with a role is delegating his role to another user till he comes back from vacation. Here delerolepre1, delerolepre2 and delerolepre3 are the postconditions which satisfy the delegation condition 'a' specified in section 2.4. The delerolepre4 ensures that the delegatee has not been already assigned this role. The delerolepre5 precondition restricts the assignment of the role to that user who is the member of same research groups as the delegator. The delegation condition 'c' is satisfied by delerolepre6 precondition which guarantees that the delegation level of delegatee does not exceed the delegation level of the delegator.

The postconditions delerolepost1 and delerolepost3 make certain that the role is delegated and the delegation level of the delegator is decremented respectively.

Now we have two types of revocation, cascading and non-cascading. For non-cascading revocation we can specify certain pre- and postconditions in the method revokerole_noncascade() as shown in Fig.4. As discussed in section 2.4.2, revokerolenpre1 satisfies condition 'a' of non-cascading revocation that the revoker must have the right to revoke. The condition 'b' is satisfied by revokerolenpre2 and revokerolepre3 by applying restrictions on the revoker such as he must belong to the same research group as delegatee and his designation must be 'professor'. The condition 'c' can be assured by not disturbing the delegations made by the delegatee. The postcondition revokerolepost1 ensures that the role is revoked.

```
User::revokerole_noncascade(revoked:User, rolename:Role)

pre revokerolenpre1: self.role_O->includes(rolename)      Non-cascading revocation condition 'a'

pre revokerolenpre2: self.group=revoked.group
pre revokerolenpre3: self.designation='prof'              Non-cascading revocation condition 'b'

pre revokerolenpre4: revoked.role_D->includes(rolename)

post revokerolenpost1: revoked.role_D->excludes(rolename)
```

Figure4 Pre-Postconditions of Noncascade Revocation Method

For cascading revocation, we have the method named revokerole_cascade()shown in Fig.5, containing the revokerolecpre1 precondition satisfying cascading revocation condition 'a' of section 2.4.3. The condition 'b' is satisfied by revokerolecpre2 and revokerolecpre3. The postcondition revokerolecpost1 ensures that the role is revoked and revokerolecpost2 makes it certain that all delegations which are dependent on the revoked delegation are also removed by checking them with the exists() operation.

```
context User::revokerole_cascade(revoked:User, rolename:Role)

pre revokerolecpre1: self.role_O->includes(rolename)      Cascading revocation condition 'a'

pre revokerolecpre2: self.group=revoked.group
pre revokerolecpre3: self.designation='prof'              Cascading revocation condition 'b'

pre revokerolecpre4: revoked.role_D->includes(rolename)

post revokerolecpost1: revoked.role_D->excludes(rolename)
post revokerolecpost2: not(Role_Assignment.allInstances->
exists(ra| ra.delegatorname=revoked))
```

Figure5 Pre-Postconditions of Cascade Revocation Method

The condition 'd' of delegation and non-cascading revocation along with condition 'c' of cascading revocation which formulates time based delegation and revocation of role assignment can be checked by creating a dynamic invariant on the delegator which checks the vacation status of the delegator as shown in Fig.6. Dynamic invariants are those invariants which are created at runtime on the objects of the classes.

```
context User inv check_vaca_status:
let u1:User=User.allInstances->any(id='mohsin')
in u1.vaca_status=true
```

Figure6 A dynamic invariant on object of User class

By adding a dynamic invariant here, we are able to restrict the delegatee to retain the assigned role until the delegator with id='mohsin' is on vacations. In the same way we can add some more dynamic invariants on the delegator's work load limit so as to ensure that the delegator's role is temporarily delegated to the delegatee as shown in Fig.7. These invariants are added with the help of the addClassInvariant() method of the GModel class of the USE API to ensure that the delegation is temporary and on failure of such constraints the revocation process occurs which also follows some pre- and post conditions

```
context Role_Assignment inv check_workload_limit:
let ra1:Role_Assignment=Role_Assignment.allInstances->(username='mohsin')
in ra1.work_load_exceed=false
```

Figure7 A dynamic invariant on object of Role_Assignment class

The method of adding dynamic invariants in the model, helps to delegate the rights to other users temporary as this can only be the mechanism to delegate a role to another user for a specific period of time following certain conditions.

B. Validation of Delegation and Revocation constraints with Pre- and Postconditions

For validation of delegation/revocation constraints we have the USE API which can be employed to load the model of the class diagram discussed in Section 3.1 containing User, Role, Research_Group and Role_Assignment classes and then using the methods of delegation and revocation. After loading the model in USE, we execute state manipulation commands for the creation of objects. The UML object diagram in Fig. 8 shows User1 and User2 belonging to Research_Group1. User1 is the original user of Role1. Role_Assignment1 contains the information regarding User1 such as the Role assigned to him as well as the delegation level of that role which ensures that User1 cannot delegate Role1 to more than three users.

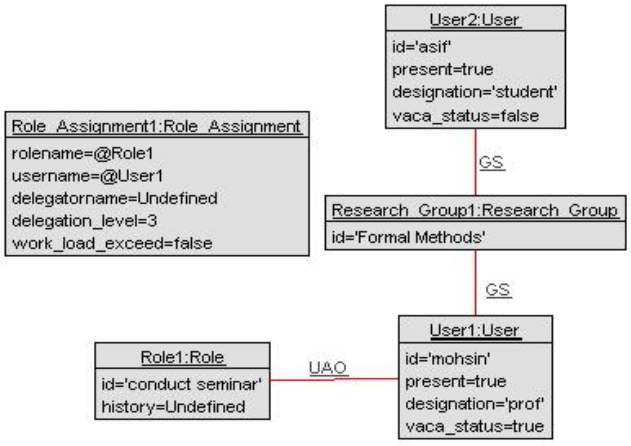

Figure8 Object Diagram of current Scenario

In order to assign Role1 to User2 we have called upon the delegaterole_D() method of the User class which has certain pre- and postconditions, specified in section 3.1. These pre- and postconditions must be satisfied for the role to be delegated to User2. After delegation of role Role1 to User2 a dynamic invariant is added to check the vacation status of User1.

The Fig. 9 shows User1 who has delegated his Role1 to User2 belonging to the same research group until he returns from vacation. The vacation status of User1 is checked here by the OCL expression evaluator showing that User1 has

returned from vacation such that the UAD made by User1 must be revoked. The model is validated before and after the delegation of roles to users by automatic snapshot generation facility of USE tool.

The validation mechanism discussed here ensures that the role is delegated temporary following the conditions, which must be satisfied before delegating the role to another user. The USE tool API helps inscribing the delegation and revocation constraints using pre- and postconditions in OCL.

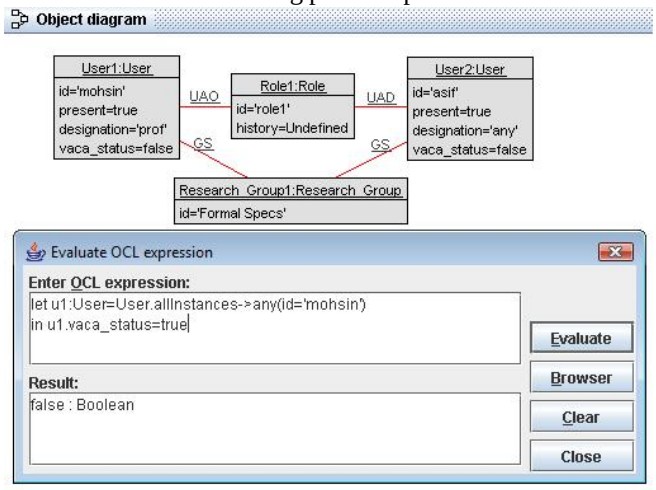

Figure9 Checking invariants with an OCL expression

Furthermore, the dynamic invariant addition facility of USE tool makes it possible to delegate and revoke role to and from the user at specific time intervals.

IV. Related Work

There has been a significant work on the delegation and revocation of roles and different models have been proposed showing Delegation of rights to users on temporal basis such as [3], [4], [10], [12].

The weakness of the permission based delegation models proposed by Zhang et al. [3] is that they give a centralized approach where a security officer is responsible for delegation and revocation of rights to users. The delegation model in [4] is very interesting as it supports direct role delegation and attribute-based role delegation along with the capability of defining maximum delegation depth that a delegatee can further delegate a delegated role. However, they have not specified the restrictions for delegation of a role in a formal language so that they can be validated before the delegation occurs.

In [10], Wainer et al. discusses the delegation mechanism in a well mannered way but when comes across temporal delegation of roles then the model only deals with the fixed time periods and leaves the issue of those temporal delegations where the exact time period of delegation is not known. Similarly in [12], Zhang models a delegation mechanism with fixed time duration and doesn't converse on temporal delegations with variable time durations.

The framework by Barka and Sandhu [14] discusses various issues regarding delegation and revocation of roles but lacks multilevel delegation if the role is temporary delegated to a user.

V. Conclusion And Future Work

The proposed validation mechanism provides flexible delegation and revocation of job roles to and from users with specific conditions. Along with validation, we attempt to specify and validate n-level delegation and cascading/non-cascading revocation processes in an organization following the delegation revocation model discussed by Nguyen et al. [4] in section 2.4 of this paper. The USE tool API is used for the validation of pre- and postconditions of delegation and revocation methods and dynamic invariants are added to formulate the job assignments as temporary. We believe this work will help understand the validation of temporary delegation and revocation of roles with UML and OCL.

There is still a lot of work required in this field of research. Other delegation and revocation models may also be validated with the help of this mechanism as well as USE tool can be further extended by creating a graphical user interface for delegation and revocation of rights temporary.

Acknowledgment

The authors gratefully acknowledge Prof. Dr. A.Q.K. Rajput, Vice Chancellor, Mehran UET, Jamshoro, Pakistan for his kind support and guidance in this research work.

References

[1] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," ACM Trans. Information and System Security, vol. 4, no. 3, pp. 224-274, Aug. 2001.

[2] M. Richters, "A Precise Approach to Validating UML Models and OCL Constraints," Ph.D. dissertation, Universitat Bremen, Fachbereich Mathematik und Informatik, Logos Verlag, Berlin, BISS Monographs, No. 14, 2002.

[3] Zhang, X., Oh, S., and Sandhu, R. 2003. PBDM: a flexible delegation model in RBAC. In Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (Como, Italy, June 02 - 03, 2003). SACMAT '03. ACM, New York, NY, 149-157.

[4] Nguyen, T., Su, L., Inman, G., and Chadwick, D. 2007. Flexible and Manageable Delegation of Authority in RBAC. In Proceedings of the 21st international Conference on Advanced information Networking and Applications Workshops - Volume 02 (May 21 - 23, 2007). AINAW. IEEE Computer Society, Washington, DC, 453-458.

[5] Wei Qiu, Carlisle Adams, "Exploring User-to-Role Delegation in Role-Based Access Control," wcmeb, p. 21, Eighth World Congress on the Management of eBusiness (WCMeB 2007), 2007.

[6] http://www.um.es/giisw/ocltools/

[7] M. Richters. The USE tool: A UML-based specification environment, 2001.Internet: http://www.db.informatik.uni-bremen.de/projects/USE/. 20, 133, 147.

[8] Sohr, K., Ahn, G., and Migge, L. 2005. Articulating and enforcing authorisation policies with UML and OCL. In Proceedings of the 2005 Workshop on Software Engineering For Secure Systems&Mdash;Building Trustworthy Applications (St. Louis, Missouri, May 15 - 16, 2005). SESS '05. ACM, New York, NY, 1-7.

[9] Karsten Sohr, Michael Drouineaud, Gail-Joon Ahn, Martin Gogolla, "Analyzing and Managing Role-Based Access Control Policies," IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 7, pp. 924-939, Jul., 2008.

[10] Wainer, J. and Kumar, A. 2005. A fine-grained, controllable, user-to-user delegation method in RBAC. In Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies (Stockholm, Sweden, June 01 - 03, 2005). SACMAT '05. ACM, New York, NY, 59-66.

[11] L. Zhang, G. J. Ahn, and B. T. Chu. A rule-based framework for role-based delegation and revocation. ACM Transactions on Information and System Security, 6(3):404–441, 2003.
[12] Zhang, L., Ahn, G., and Chu, B. 2002. A role-based delegation framework for healthcare information systems. In Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies (Monterey, California, USA, June 03 - 04, 2002). SACMAT '02. ACM, New York, NY, 125-134.
[13] D.F. Ferraiolo and D.R. Kuhn (1992) "Role Based Access Control" 15th National Computer Security Conference, Oct, 1992.
[14] Barka, E. and Sandhu, R. 2000. Framework for role-based delegation models. In Proceedings of the 16th Annual Computer Security Applications Conference (December 11 - 15, 2000). ACSAC. IEEE Computer Society, Washington, DC, 168.

Mohsin Ali Memon was born in Hyderabad, Pakistan on 25th July, 1983. He received his B.E. (Software Engineering) in 2005 and M.E. (Information Technology) in 2009 from Mehran University of Engineering & Technology, Jamshoro, Pakistan.

He is working as LECTURER in the Department of Computer Systems and Software Engineering, Mehran University of Engineering & Technology, Jamshoro, Pakistan since 2006. His research interest includes formal methods, Software Verification techniques, Ubiquitous Computing and Interactive Multimedia.

Mr. Memon is a member of International Association of Computer Science and Information Technology (IACSIT) and Pakistan Engineering Council.

Prof. Dr. Manzoor Hashmani was born in Hyderabad, Pakistan on 6th March, 1967. He received his B.E. (Computer Systems Engineering) from Mehran University of Engineering & Technology, Jamshoro (Pakistan) in 1991, M.E in 1997 and Ph.D. in 1999 from Nara Institute of Science & Technology, Nara (Japan).

He is working as FOREIGN PROFESSOR in the Department of Computer Systems and Software Engineering, Mehran University of Engineering & Technology, Jamshoro, Pakistan. He has authored and co-authored more than 30 research papers published in various journals and conferences of international repute. He has also worked as lead research and development person in a reputable Japanese company for five years. His research areas of interest include High Speed Communication Networks, Software Engineering and Alternative Engery.

Dr. Hashmani is a member of IEICE (Japan), IEEE Communications Society (USA), and Pakistan Engineering Council.

Dr. Karsten Sohr received the doctorate degree from Universita ¨t Marburg, Germany. He works in the Center for Computing Technologies (TZI), Universita ¨t Bremen, Germany, where he is currently a coordinator for the security research area. His research interests include role-based access control and security of mobile applications. He received research grants from the German Federal Ministry of Education and Research (BMBF) and from the German Research Foundation (DFG).