Role inheritance with object-based DSD

Muhammad Asif Habib

FIM, Johannes Kepler University,
Altenbergerstraße 69, A-4040 Linz, Austria
E-mail: habib@fim.uni-linz.ac.at

Abstract: Role-based access control (RBAC) is an evolution in the field of access control. RBAC offers tight security of information and ease of management to implement. The focus of this paper is on some of the important factors in RBAC, i.e., dynamic separation of duty (DSD) which is implemented to avoid internal security threats and role inheritance. We discuss DSD from a different perspective, i.e., object-based dynamic separation of duty. Also, we discuss permission level inheritance from object perspective. Different problems and observations have been described regarding DSD with respect to formal definitions of DSD. Those observations and problems influenced us to go for updated definition of DSD. So, we propose a definition of DSD from different perspective and elaborate the importance of role inheritance. Different examples have been given regarding object-based DSD with different scenarios. We also describe the benefits of implementing the proposed definition of DSD.

Keywords: role-based access control; RBAC; role inheritance; object-based DSD; permission level inheritance.

Reference to this paper should be made as follows: Habib, M.A. (2011) 'Role inheritance with object-based DSD', *Int. J. Internet Technology and Secured Transactions*, Vol. 3, No. 2, pp.149–160.

Biographical notes: Muhammad Asif Habib received his MSc in Computer Science from the University of Punjab Lahore, Pakistan in 2004. He joined a research group at FIM, Johannes Kepler University Linz Austria in February 2008. His major research interests are in the area of information and network security, particularly access control, privacy and grid securities.

1   Introduction

The success of a business depends on the ability and strength to protect the valuable information and data. Information and data are the most precious and valuable things for any organisation. The organisations demand for fool proof security for their data and information as well as they demand for effective execution of business tasks after the implementation of a security policy. The organisations do not want the business processes to be disturbed after implementing any security policy. They do not want any such security policy that after implementing the policy, the organisations have to suffer for delay in execution of business processes or it slows down business tasks. So, the organisations demand for secure as well as efficient systems.

RBAC is known as the evolution in the field of access control. RBAC is a proven and open ended technology that is being attracted by most of the organisations for its capability to reduce security administration in terms of cost and complexity. There are three main entities which are users, roles and permissions in RBAC. The users and permissions are assigned to roles. The permissions are comprised of operations and objects (Sandhu et al., 1996). The basic story revolves around the roles. The concept of least privilege and separation of duty have given the opportunity to make the information and data more secure than before. According to ASCAA principles for next-generation role-based access control (RBAC) (Sandhu and Bhamidipati, 2008) granting and revoking the roles will be performed dynamically in future. A detailed mechanism is given in Mühlbacher and Praher (2009) about the dynamic activation and deactivation of roles. So, dynamically granting and revoking roles can be implemented efficiently after the implementation of dynamic separation of duty (DSD) in RBAC.

The definition of DSD has been given in American National Standard for Information Technology – Role-Based Access Control, ANSI INCITS (2004) as "A user can be authorised for multiple mutually exclusive roles (MER) and the user can exercise these roles independently but not at the same time or simultaneously". In this paper, we propose a different definition of DSD. We consider object as another important entity like roles, users and permissions in RBAC. In this paper, it is assumed that object can be any resource like file, folder, printer or directory, etc. The importance of implementing DSD from object perspective is highlighted. Separation of duty and mutual exclusion are very much closely related to each other. For efficient implementation of separation of duty, we need to implement sound mechanism for role inheritance. We will not be able to achieve the goal to get all benefits of implementing separation of duty in RBAC without efficient implementation of role hierarchy. So, we need to specify the roles behaviour with respect to mutual exclusion.

This paper is divided into different sections. In Section 2, the overview and background of separation of duty, its types and role inheritance are discussed at a glance. In Section 3, the observations part which contains some important points regarding role inheritance and the influence which we get to go for updated definition of DSD is discussed. In Section 4, we discuss object-based separation of duty, i.e., one part of the core of this paper. In Section 5, we discuss mutual exclusion with role inheritance with different angles. In Section 6, a number of examples of different possibilities of role activation with respect to different scenarios are discussed. At last but not least, we tried to conclude the paper.

2   Overview and background

RBAC is getting fame day by day and is being implemented in many organisations. In RBAC, some of the important factors are the principle of least privilege and separation of duty (Simon and Zurko, 1997; Giuri and Iglio, 1996). According to the concept of separation of duty, a business process or task is divided into more than one sub process or sub task. These sub tasks are assigned to different roles and different users are assigned to these roles. These roles are declared mutually exclusive to each other, i.e., these roles will not be activated by a single user at the same time ANSI INCITS (2004). So there will be a least chance of fraud if a business task is involved with more than one user.

Consequently, one user would not be able to perform one complete business task independently. The concept of role inheritance plays an important and vital role in the implementation of mutual exclusion with respect to separation of duty. If a role is inherited another role then it means that the set of permissions of one role are being contained by another role. Now, it depends that whether it is full containment or partial containment. In case of full containment, all permissions of one role are contained by other role and in case of partial containment some of the permissions are contained by other role (Lan-Sheng et al., 2006).

The concept of separation of duty has a long history. It is used to extend the liability and obligation to more than one person to minimise the chance of fraud (Simon and Zurko, 1997). The separation of duty is a security parameter used to enforce security of a management information system. The concept of separation of duty has been already discussed in different research papers (Gligor et al., 1998; Crampton, 2003; Simon and Zurko, 1997; Sandhu, 1988). To minimise the chances of fraud, a business task is needed to divide into multiple sub tasks executed by more than one role. If a complete business task is executed by only one role then there are many more chances of fraud as compared to the situation where more than one role is involved in the execution of a business task. Different forms of separation of duty have been discussed in Crampton (2003), Simon and Zurko (1997) Nash and Poland (1990), and Ferraiolo et al. (1995) as static separation of duty, DSD, object-based separation of duty, operational separation of duty and history-based separation of duty.

Suppose if a single role has permissions to create, sign and finally, approve the purchase order in a purchasing department then there will be a maximum chance of fraud, because only one person is involved in a complete business process. So nothing can stop a user who is authorised to have this role from exercising all the permissions and executing the whole business process which is against the concept of separation of duty. But if we break up this role into different sub roles representing different business sub tasks and these sub tasks executed by different roles and all these roles are constrained not to attain by a single user then there will be a least chance of fraud as compared to the previous situation. RBAC is a mechanism that is used to implement many policies but separation of duty is closely attached to RBAC because RBAC is considered as a natural mechanism for the implementation of separation of duty (Kuhn, 1997).

3 Observations

According to the definition given in ANSI INCITS (2004) about DSD, one user can not activate more than one MER at the same time. If any user wants to activate another MER then the user has to quit its previously activated MER which means a user can have multiple MER s but not at the same time. We propose updated definition of DSD due to below observations that might be helpful for the proper implementation of DSD in the real spirit.

If a user is assigned to two MER and the user has activated one of those roles for one object then logically the user should be allowed to activate other MER for other object at the same time. It does not make sense if a user is not allowed to activate a MER for one object even though the user is authorised to activate that role and also no role is activated for that object before. The activation of a MER for a certain object does matter on the

same pattern as it does matter which user has activated which role as "who performed which step on which object" given in Schaad et al. (2005).

According to the definition of DSD given in ANSI INCITS (2004), one user can be assigned to all MER and the user can activate only one of the MER at the same time. Suppose if the user wants to switch from one MER to another then it can be done only by quitting the previously activated role. This is important to note that if one business task is divided into different subtasks which are executed by different MER and one user is assigned to all those MER then one user can activate all roles one by one at different time and execute the whole business task. This is against the concept of separation of duty. So, the user can be authorised to all MER but it should not be allowed that a user can activate all MER for the same object.

The maximum number of activated roles by a user for the same object should always be less than the total number of roles comprised of a complete business process. This is an important parameter which is called cardinality (Tidswell, and Jaeger, 2000). A user should not be allowed to activate two successive dependent roles for the same object. If one role depends on other role and a user is authorised for both of these roles then user should be restricted to activate only one role for the same object.

All the above observations are discussed in greater detail in next section.

4   Object-based DSD

The proposed definition of DSD is given after getting influence from above observations which is contrary to the definition given in ANSI INCITS (2004) as "A user can be assigned to all MER and is allowed to activate all MER at the same time but not for the same object and also the user is not allowed to activate two successive dependent roles for the same object". So, it means a user will be allowed to activate MER for different objects at the same time. Also, a user will not be allowed to activate all MER for the same object neither at same time nor at different time. After implementing the proposed definition of DSD, there will be no waste of resources because if a user wants to activate another MER for other object, then that user will be allowed to do that (Habib and Praher, 2009). Also, no user will be allowed to execute the whole business task by activating all MER at different times. So, there will be proper implementation of the concept of separation of duty.

The implementation of newly proposed definition of DSD depends on the proper implementation of role hierarchy, history-based separation of duty and operational separation of duty (Crampton, 2003; Simon and Zurko, 1997; Nash and Poland, 1990; Ferraiolo et al., 1995). But we did not discuss these stated concepts in detail.

Nash and Poland proposed a rule in Nash and Poland (1990) about object-based DSD. In this rule, they proposed that a user can execute a transaction if the user is authorised to execute that transaction and the user has not executed any other transaction on that object or data item before. We agree to this rule because it supports a part of our proposed definition of DSD. There are some points left for discussion like there may be a maximum limit for a user to activate a specific number of MER. A user can activate a role for an object if the user is authorised to activate that role and user has not reached the maximum limit of activated MER.

While discussing an example, we will refer the reader to the example given in Nash and Poland (1990), there are three roles like enter, verify and authorise. Also there are three users like clerk, officer and supervisor authorised for above roles sequentially. All these roles are mutually exclusive to each other. As we discussed earlier that object-based DSD is a composite concept which requires the proper implementation of role hierarchy, history-based separation of duty and operational separation of duty. The role 'enter' is assigned to clerk, the role 'verify' is assigned to officer and the role 'authorise' is assigned to supervisor. As per operational separation of duty, a user is not allowed to activate all MER for the same object neither at the same time nor at other time. In this way, one user will not be able to execute the whole business process itself (Ferraiolo et al., 1995). According to the role hierarchy 'clerk' is the junior most user assigned to role 'enter', officer is at the middle level authorised for both roles 'enter' and 'verify' and at last the supervisor is the senior most user authorised for all roles 'enter', 'verify' and 'authorise'.

According to our proposal the user is not allowed to activate two successive dependent roles for the same object. For instance, in this example the officer is authorised for the roles 'enter' and 'verify'. The role 'verify' depends on the role 'enter'. So, if the officer activates role 'enter' then the other dependent role 'verify' should not be allowed to be activated by the officer for the same object. Because it does not make sense that a user verifies its own action. On the same pattern, the supervisor is authorised to activate all three roles but the supervisor should not be allowed to activate successive dependent roles for the same object. In this example, the supervisor is either allowed to activate the roles 'enter' and 'authorise' for the same object. If the supervisor activates the role 'verify' then the supervisor should not be allowed to activate the roles 'enter' and 'authorise'. So there should be proper record keeping of the roles activated by the user for the object (Sandhu, 1988) called the history-based separation of duty (Simon and Zurko, 1997).

5 Mutual exclusion with role inheritance

The affective implementation of separation of duty requires the affective implementation of mutual exclusion of roles. When we go for the implementation of mutual exclusion of roles then we can not afford to neglect the role of inheritance of roles. RBAC is a modular technology which can be implemented in modules as per requirements of the organisation. We can have two methods of role inheritance. First will be role level inheritance and second will be permission level role inheritance.

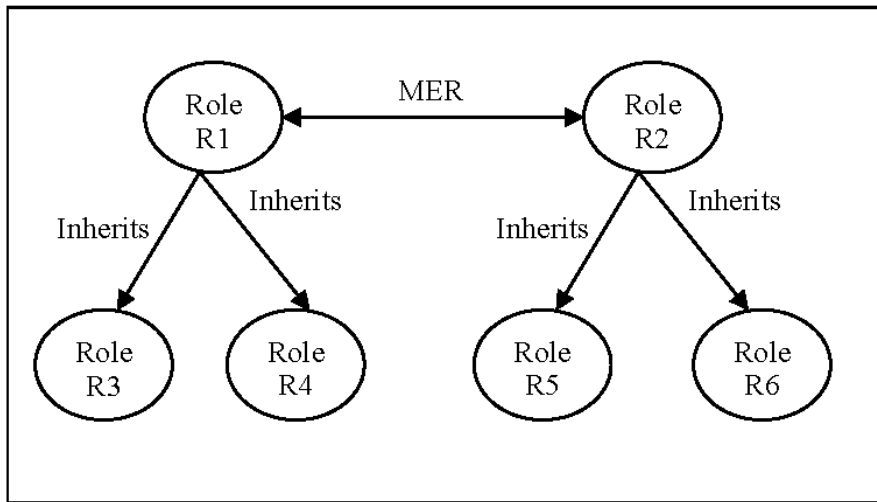### 5.1 Mutual exclusion with role level inheritance

In case of role level inheritance, we will not go deeper at the level of permissions. We will remain on the level of roles only. When one role is made mutually exclusive with another role, then it means all the permissions of one role are made mutually exclusive with the permissions of other role. So, a role will be fully mutually exclusive with other role.

In Figure 1, we have two MER R1 and R2. It means that these two roles can not be activated in case of dynamic mutual exclusion. Role R1 inherits further two more roles R3 and R4 as well as Role R2 inherits two further roles R4 and R5. Suppose role R1 has three permissions P1, P2 and P3. On the same pattern Role R2 has three permissions P4, P5 and P6. The role R1 inherits roles R3 and R4 which have permissions P1 and P2 respectively.

$$R1 = \{P1, P2, P3\} \qquad R2 = \{P4, P5, P6\}$$

$$R3 = \{P1\} \qquad R5 = \{P4\}$$

$$R4 = \{P2\} \qquad R6 = \{P5\}$$

On the same pattern, the role R2 inherits role R5 and R6 which have permissions P4 and P5. If we want to implement real mutual exclusion of roles R1 and R2 then we have to maintain the mutual exclusion of the inherited roles. So, keeping the above scenario in mind, the role R5 should be mutually exclusive to role R3 and R4 and on the same pattern role R6 should be mutually exclusive to the roles R3 and R4. If we would be able implement mutual exclusion as per above statement then there will be real and dynamic implementation of separation of duty because implementation of separation of duty requires the implementation of mutual exclusion of roles. When the roles R1 and R2 will be made mutually exclusive to each other then the inherited roles or contained roles R3 and R4 will be made mutually exclusive to R5 and R6 automatically without the intervention of the security administrator. We strongly recommend the proper and dynamic implementation of mutual exclusion of roles for the implementation of DSD.

Figure 1   Mutual exclusion with role inheritance

5.2   *Mutual exclusion with permission level inheritance*
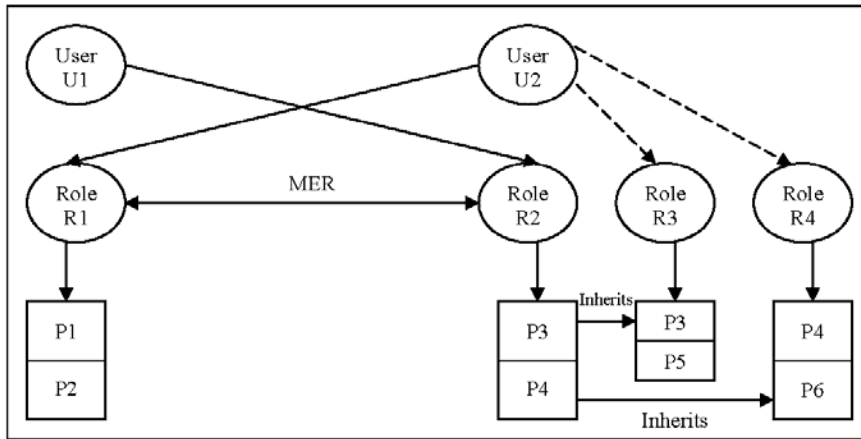
We have already explained that there are two ways or methods of implementing mutual exclusion of roles with respect to inheritance. First is role level inheritance and second

is deeper level of inheritance which is permission level inheritance. In latter case permissions itself are treated as roles. The behaviour of the permissions will be like roles as it was in above case.

In Figure 2, we have two users U1 and U2. We have two MER R1 and R2. Both users have been assigned for both roles but no user can exercise both roles at the same time due to mutual exclusion of R1 and R2. Role R1 is comprised of permissions P1 and P2.

On the same pattern role R2 has two permissions P3 and P4. User U1 exercises role R2 and user U2 exercises role R1. If we will go deeper then we will come to know that role R2 inherits role R3 and role R4. Role R3 and role R4 have their own permissions as well as the inherited permissions from their parent roles. Role R3 has permissions P3 and P5 as well as role R4 has permissions P4 and P6.

Figure 2 Mutual exclusion with role inheritance

The arrows from user U1 to role R2 and from user U2 to role R1 are showing that the users are exercising these roles. But arrows from user U2 to role R3 and role R4 are showing that the user U2 can exercise these roles if there will not be a proper implementation of the role inheritance.

When the roles R1 and R2 are made mutually exclusive to each other then their inherited or contained roles should also be made mutually exclusive to each other automatically without the intervention of the security administrator. If we will not take care of this problem then the user U2 can exercise roles R3 and R4 by treating them as separate and independent roles. In this way, the roles R1 and R2 will not be MER any more. As a result of this process, DSD will not be implemented, which can cause the frauds to commit due to one man control.

So, for the proper implementation of DSD, we should implement role inheritance in a proper and dynamic way.

5.3 Mutual exclusion with object-based permission level inheritance

We have elaborated two different scenarios above relating to role inheritance. In above scenarios, we did not discuss the impact of objects in the system. In the current scenario, we will discuss the role inheritance in the presence of objects role in the system.
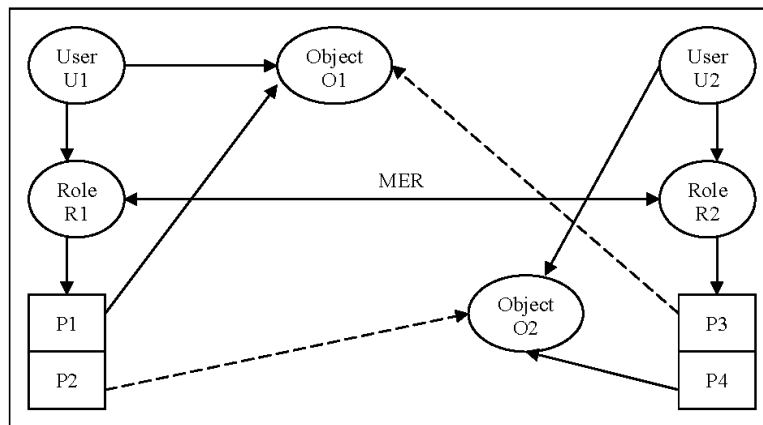
In this scenario, there are two users U1 and U2. There are two roles R1 and R2 which are mutually exclusive to each other. Role R1 has two permissions P1 and P2 and role R2 has also two permissions P3 and P4. There are two objects O1 and O2 involved in this system.

The user U1 is activating role R1 and user U2 is activating role R2. User U1 is activating a part of the permissions of Role R1 which is P1 for object O1. On the same pattern, the user U2 is activating a part of the permissions for the object O2 which is P4.

The arrow from P4 to object O2 is showing that user U2 is activating the permission P4 of the role R2 for the object O2. On the same pattern the arrow from P1 to object O1 is showing that the user U1 is activating permission P1 of the role R1 for the object O1. If the picture of the inheritance of the roles will not be clear then it will be difficult to decide whether the user can activate other permissions of the role but for the different object.

So, all given scenarios showing that if we want to make sure the proper and dynamic implementation of separation of duty then we have to make sure the proper implementation of MER which requires the implementation of role inheritance.

Figure 3   Mutual exclusion with object-based role inheritance

6   Scenarios

Different examples have been described where we differentiate the after affects of implementing proposed definition of DSD and the definition given in ANSI INCITS (2004). Also, we try to elaborate the effectiveness and practical as well as dynamic nature of proposed definition.

6.1   Case 1

In this case, we suppose that there are two MER R1 and R2. Role R1 and R2 are comprised of one complete business task. User U1 is already assigned to both roles. In Figure 4, the user U1 has activated role R1 for object O1.

Suppose we follow the definition of DSD given in ANSI INCITS (2004), according to the Figure 4, if the user U1 wants to activate role R2 for the same object O1 then the role R2 can be activated by quitting role R1 and activating role R2. In this way, the user can execute the whole business task which is against the concept of separation of duty.

If we follow the proposed definition of DSD then the user U1 can not activate role R2 at same time or at some other time shown in Table 1. According to the proposed definition of DSD, the concept of separation of duty will be implemented properly. In Table 1, the activation of role R1 to user U1 for object O1 is granted but the activation of role R2 to user U1 for object O1 is denied.

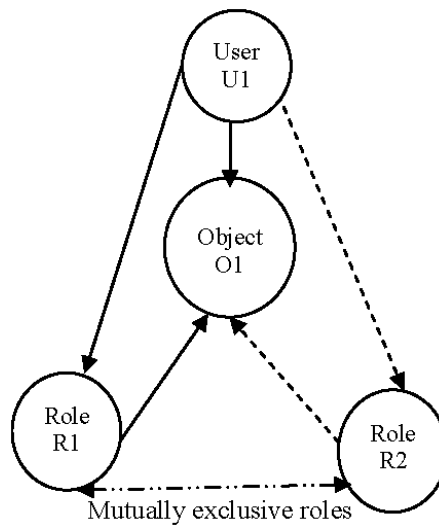**Figure 4**   Single user/object with multiple roles

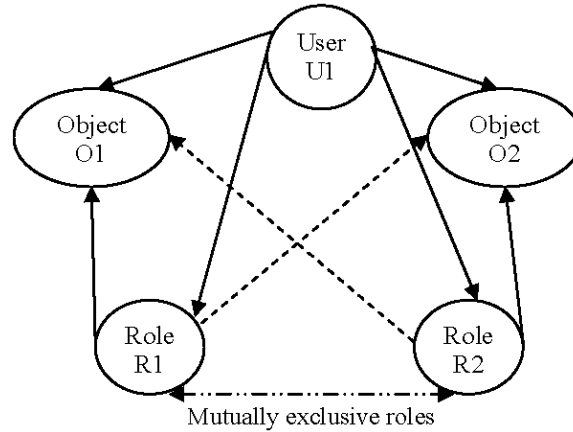**Table 1**   Multiple roles activation to single user with single object

Role	User	Object	Activation
R1	U1	O1	Granted
R2	U1	O1	Denied

6.2   Case 2

In this case, we suppose that there is one user U1 who is assigned to two MER R1 and R2 which make one complete business task. There are two objects involved in this case also. The user U1 is already assigned to both roles. The user U1 has activated role R1 for object O1.

Suppose we follow the definition of DSD given in ANSI INCITS (2004) then if the user U1 wants to activate role R2 for the object O2 at the same time, then the user U1 will not be allowed to activate role R2 for object O2 because roles R1 and R2 are mutually exclusive to each other that is why they can not be activated by the same user at the same time as shown in Figure 5.

Figure 5   Single user with multiple objects/roles

Also, if the user U1 wants to activate role R2 for the same object O1, then the user U1 will quit its previously activated role R1 and will activate role R2 for the object O1. By following the proposed definition of DSD, if the user U1 wants to activate the role R2 for other object O2, then the user will be able to activate that role as shown in Table 2.

Also, if the user wants to activate role R2 for the same object O1, then it will not be allowed to activate this role for the object. So, we believe that this is a practical and dynamic approach.

Table 2   Multiple roles activation to single user with multiple objects

Role	User	Object	Activation
R1	U1	O1	Granted
R2	U1	O2	Granted
R2	U1	O1	Denied
R1	U1	O2	Denied

6.3   Case 3

In this case, multiple users, roles and objects are involved. We suppose that there are two users U1 and U2 are assigned to MER R1 and R2 which make one complete business task. There are three objects O1, O2 and O3 involved in this case also. The user U1 has already activated role R1 for object O1 and the user U2 has activated role R2 for object O2.

Suppose we follow the definition of DSD given in ANSI INCITS (2004), given that if the user U2 wants to activate role R1 for an other object O3 at the same time then the user U2 will not be allowed to activate role R1 for object O3 because both roles R1 and R2 are MER and can not be activated by the user U2 at the same time. Also, if the user U1 wants to activate role R2 for the same object O1, the user U1 can do this by quitting its previously activated role R1 and activating new role R2. On the same pattern, if the user U2 wants to activate role R1 for same object O2, the user U2 can do this by quitting its previously activated role R2 and activating new role R1.

If we follow the proposed definition of DSD given that if the user U2 wants to activate role R1 for object O3, then the user U2 will be allowed to do this as shown in Table 3. If the user U1 wants to activate role R2 for object O1 then the user U1 will not be allowed to do that. Also on the same pattern if the user U2 wants to activate role R1 for the object O2, then the user U2 will not be allowed to do that.

According to the previously discussed cases we have tried to emphasise that we can afford to implement the concept of separation of duty by implementing the definition of DSD given in ANSI INCITS (2004). We can implement the real spirit of the concept of separation of duty with the proposed definition of DSD.

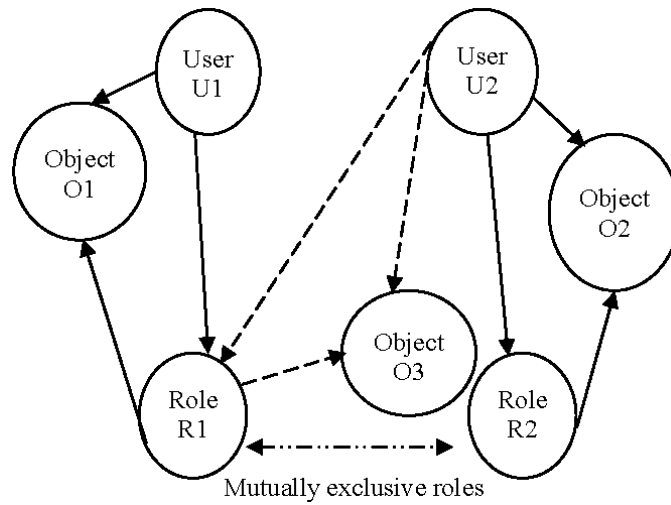**Figure 6** Multiple users with multiple objects/roles

Table 3 Multiple roles activation to single user with multiple objects

Role	User	Object	Activation
R1	U1	O1	Granted
R2	U2	O2	Granted
R1	U2	O3	Granted
R1	U2	O2	Denied

7 Conclusions

The implementation of RBAC with proposed definition of DSD can resolve many problems which we discussed in the observations section. Also for proper and dynamic implementation of separation of duty we must have to implement real and dynamic role inheritance. No single user will be allowed to execute the whole business process. The resources can be maximally utilised and the concept of separation of duty will be implemented in a better way in the form of effectiveness and dynamicity. This will be more dynamic and practical approach as compared to formal definition of DSD given in

ANSI INCITS (2004). We are currently in the phase of implementing the proposed definition of DSD.

References

American National Standard for Information Technology – Role Based Access Control (2004) ANSI INCITS 359-2004.

Crampton, J. (2003) 'Specifying and enforcing constraints in role-based access control', in *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies (Como, Italy, 2–3 June 2003). SACMAT '03*, pp.43–50, ACM, New York, NY, DOI = http://doi.acm.org/10.1145/775412.775419.

Ferraiolo, D., Cugini, J. and Kuhn, D.R. (1995) 'Role-based access control (RBAC): features and motivations', *Proc. 1995 Computer Security Applications Conference*, December, pp.241–248.

Giuri, L. and Iglio, P. (1996) 'A formal model for role-based access control with constraints', *Ninth IEEE Computer Security Foundations Workshop, CSFW*, p.136.

Gligor, V., Gavrila, S. and Ferraiolo, D. (1998) 'On the formal definition of separation-of-duty policies and their composition', in *Proceedings of 1998 IEEE Symposium on Research in Security and Privacy*, pp.172–183, Oakland, California.

Habib, M.A. and Praher, C. (2009) 'Object based dynamic separation of duty in RBAC', *Internet Technology and Secured Transactions, 2009. ICITST 2009.*, 9–12 November, pp.512–516.

Kuhn, D.R. (1997) 'Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems', in *Proceedings of the Second ACM Workshop on Role-Based Access Control (Fairfax, Virginia, USA, 6–7 November 1997). RBAC '97*, pp.23–30, ACM, New York, NY, DOI = http://doi.acm.org/10.1145/266741.266749.

Lan-Sheng, H., Fan, H. and Koio, A.B. (2006) 'Least privileges and role's inheritance of RBAC [J]', *Wuhan University Journal of Natural Sciences*, No. 11.

Mühlbacher, J.R. and Praher, C (2009) 'DS RBAC – dynamic sessions in role based access control', in *Journal of Universal Computer Science*, Vol. 15, No. 3, pp.538–554, ISSN 0948-695x.

Nash, M.J. and Poland, K.R. (1990) 'Some conundrums concerning separation of duty', in *Proceedings of the Symposium on Security and Privacy*, pp.201–207.

Sandhu, R. (1988) 'Transaction control expressions for separation of duties', in *Proceedings of 4th Aerospace Computer Security Conference*, pp.282–286, Orlando, Florida.

Sandhu, R. and Bhamidipati, V. (2008) 'The ASCAA principles for next-generation role-based access control', *Proc. 3rd International Conference on Availability, Reliability and Security (ARES)*, 4–7 March, pp.27–32, Presentation Keynote Lecture, Barcelona, Spain.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (1996) 'Role-based access control models', *Computer*, February, Vol. 29, No. 2, pp.38–47, doi:10.1109/2.485845.

Schaad, A., Spadone, P. and Weichsel, H. (2005) 'A case study of separation of duty properties in the context of the Austrian 'eLaw' process', in Liebrock, L.M. (Ed.): *Proceedings of the 2005 ACM Symposium on Applied Computing, (Santa Fe, New Mexico, 13–17 March 2005), SAC '05*, pp.1328–1332, ACM, New York, NY, DOI = http://doi.acm.org/10.1145/1066677.1066976.

Simon, R. and Zurko, M. (1997) 'Separation of duty in role-based environments', in *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pp.183–194, Rockport, Massachusetts.

Tidswell, J.E. and Jaeger, T. (2000) 'Integrated constraints and inheritance in DTAC', in *Proceedings of the Fifth ACM Workshop on Role-Based Access Control*, Berlin.