Separation of Duty Administration

S Perelson[a] R Botha[a] J Eloff[b]

[a]*Faculty of Computer Studies, Port Elizabeth Technikon, Port Elizabeth*
{stephen,reinhard}@petech.ac.za
[b]*Department of Computer Science, Rand Afrikaans University, Johannesburg*
eloff@rkw.rau.ac.za

Abstract

Access control administration is a huge task. Administration tools should assist the administrator in ensuring that the access control requirements are met. One example of an access control requirement is Separation of Duty (SoD). SoD requirements specify that no single person may have sufficient authority to complete a business process unilaterally.
The SoDA prototype administration tool has been developed to assist administrators with the administration of SoD requirements. It demonstrates how the specification of both Static and Dynamic SoD requirements can be done based on the "conflicting entities" paradigm. Static SoD requirements must be enforced in the administration environment. The SoDA prototype, therefore, enforces the specified static SoD requirements.
Keywords: *Information Security, Access Control Administration, Separation of Duty*
Computing Review Categories: *D4.6, H2.7, H4.1, K6.5*

1 Introduction

Security administrators must manage an ever-increasing number of systems under their control. In recent years, Role-based Access Control (RBAC) has been promoted as a possible solution to the resultant administration nightmares [5]. With the increasing amount of information available electronically, it is necessary not only to find a means to ease the job of the security administrator, but also to ensure that the information is protected and managed according to organizational policies.

One expression of organizational policy can be found in the age-old principle of Separation of Duty (SoD). Saltzer and Schroeder [10] identified SoD, or "separation of privilege" as they called it, as one of eight design principles for the protection of information in computer systems. They built on the observation that a security system with two keys is more robust and flexible than one that requires a single key. No single accident, deception or breach of trust is therefore sufficient to compromise the system. Clark and Wilson [4] identified SoD as one of the two major mechanisms that can be implemented to ensure data integrity. SoD serves as a mechanism to counteract fraud and error, while assuring correspondence between system objects and the real world objects that they represent.

Furthermore, they [4] asserted that, at the policy level, processes are divided into tasks, with each task being performed by a different person. [1] and [8] observed that existing SoD models do not take work processes into consideration. Work processes are often facilitated through the use of workflow systems. Workflow systems are constructed around tasks that are linked according to business rules to represent a business process. This paper introduces the task as an additional building block for expressing SoD requirements in workflow systems.

Even with the introduction of the task abstraction, the administration of SoD requirements remains a mammoth task. In a large organization, there may be thousands of objects that require protection. The organization may have thousands of users, filling hundreds of different positions in the organization. The identification of all the access requirements requires a huge effort. It is virtually impossible to maintain consistency when performing such a huge task, unless the administration tools provide appropriate assistance.

The SoDA prototype is introduced to assist security administrators with the specification of access control requirements according to Role-based Access Control principles. More specifically, the SoDA prototype is intended to assist with the administration of SoD requirements. In order to demonstrate the "conflicting entities" administration paradigm as used within the SoDA prototype, the remainder of the paper is structured as follows. First, a brief review of role-based access control principles is provided. Thereafter, the additional concept of a task is introduced. This is followed by a discussion on the use of the "conflicting entities" paradigm to specify SoD requirements. Finally, we illustrate how the SoDA prototype is used to administer SoD requirements.

2 Basic Concepts

This section will provide the necessary background to explain the principle of separation of duty within role-based workflow systems.

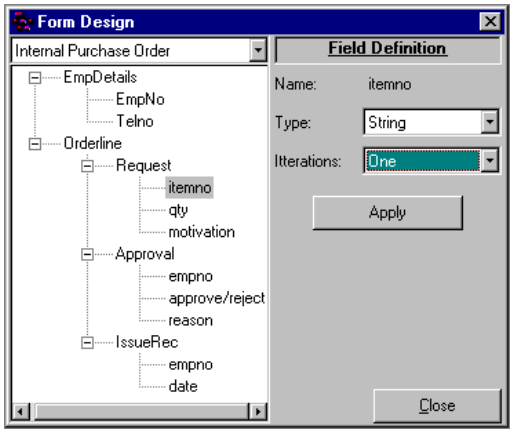

Figure 1: Form design environment used to create a "Purchase Order"

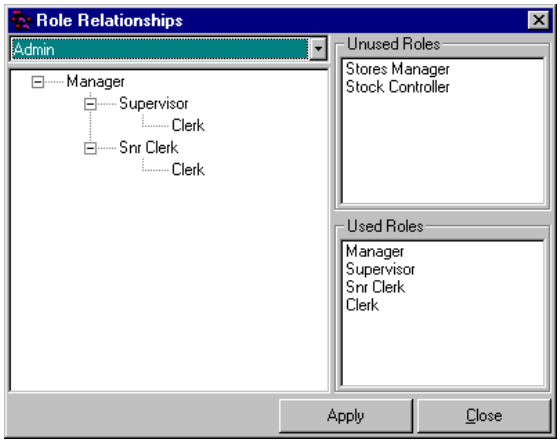

Figure 2: SoDA associates roles according to named role networks

2.1 Role-based Access Control

The concept of a role is pivotal in role-based access control. Users receive access permissions based on the roles that they may assume. Users are anyone/anything that accesses resources in the system. A user may, therefore, be an individual or another program. Roles often correspond to positions in the organizational structure. It is thus a semantic construct, created to ease the management of access rights. Permissions can be interpreted as the right to execute a certain method of an object.

The SoDA prototype considers an object to be a document containing various field objects. Users may perform different actions on the field objects, e.g. add another instance of the field object, delete a field object, edit the contents of a field object or view the contents of a field object. Individual field objects may be grouped, resulting in composite objects. Figure 1 shows how a hierarchical view, representing object containment, can be used to create the 'Internal Purchase Order' object. Permissions could relate to any of the field objects, or composite field objects, in the 'Internal Purchase Order' object. Permissions assigned to an object are inherited for objects contained by that object. For example, the permission to edit Employee Details will imply the permission to edit all fields that form part of Employee Details on the form.

Roles may be related through a partial order. A role inherits permissions assigned to the roles that are junior to it in the partial order. For example, the 'Manager' role may be considered senior to the 'Supervisor' role. The 'Manager' role will, therefore, inherit the permissions assigned to the 'Clerk' role. Figure 2 shows how the SoDA prototype manages the associations between roles. In SoDA, roles are related to other roles within disjoint, named role networks. The combination of all named role networks is similar to the role-graph presented by [8], if an artificial maximum and an artificial minimum role were introduced.

The concepts employed in RBAC are indeed very powerful. However, Sandhu et al. [11] observed that:

> "RBAC is not a panacea for all access control issues. More sophisticated methods are required to deal with situations that control operation sequences. [...] Other forms of access control can be layered on top of RBAC for this purpose."

Workflow Systems provides an environment where the sequences of operations are controlled according to business rules. The next section introduces workflow concepts, paving the way for the expression of access control policies in terms of sequence of operations.

2.2 Workflow Concepts

Workflow Systems are concerned with the automation and facilitation of business processes [6]. Business processes are defined through process definitions. A process definition consists of sets of tasks, connected according to business rules.

The process definition is enacted by the workflow engine. For each enactment of the business process, e.g. for each 'Internal Purchase Order' that is issued, a process instance is generated. Task instances are generated on demand, based on the business rules encapsulated as part of the process definition.

SoDA is a tool that focuses on supporting access control administration. Access control requirements are, typically, described within the general context of a business process and not for a specific enactment of the workflow. The SoDA prototype is, therefore, only concerned with the process and task definitions.

The "conflicting entities" paradigm relies on restricting the associations between all the entities that are involved, namely user, roles, permissions and tasks.

3 SoDA – The "conflicting entities" paradigm

Separation of duty requirements are implemented by restricting the associations allowed between entities. This is to ensure that a single user may not receive too many permissions. An example of such a constraint may specify that

"the permission to approve an order and the permission to issue an order may not be assigned to the same role".

Kuhn [7] explained how mutual exclusive roles, i.e. roles that may not be assigned to the same user, can be used to enforce SoD. Ahn and Sandhu [1] showed through their RSL99 specification language that there are several ways of expressing similar SoD requirements. SoDA builds on these observations, and extends it with the concept of conflicting tasks.

The term "conflicting entities" does not indicate that there are any disharmony between the entities. The "conflict" refer, rather to the disharmony that the entities could cause between the actual and the desired state of the system. Conflict thus indicates a potential undesirable state of integrity. The "conflicting entities" paradigm, as employed in the SoDA prototype, identifies four types of conflict [3]:

Conflicting permissions are permissions that can result in unnecessary power if bestowed on the same person. For example, a person with the permissions required for financial audits should not receive permissions to approve financial transactions. If this were allowed, auditors could lose their independence.

Conflicting users are users who will together have sufficient power to collude, and are likely to do so. In practice, this may be family members or previously known accomplices.

Conflicting roles are roles that together possess the ability to conspire. This means that they are assigned conflicting permissions. Consider, for example, the 'Auditor' and 'Financial Manager' roles. It is common practice that auditors and financial managers should be independent. The roles may have certain permissions, e.g. 'view order', in common. However, the 'approve order' and 'approve audit' permissions may be assigned only to one of these roles.

Conflicting tasks are tasks requiring conflicting permissions to complete. This would, for example, imply that the 'Audit Purchase Order' task and the 'Approve Purchase Order' task would be conflicting since they require the 'approve order' and 'approve audit' permissions. These permissions are, in turn, conflicting.

The "conflicting entities" paradigm is based on the observation that power is vested in permissions. The essence of the "conflicting entities" paradigm lies, therefore, in conflicting permissions. It is argued, however, that tasks provide a more natural abstraction for the specification of SoD requirements. The "conflicting entities" paradigm allows for the specification of both Static and Dynamic SoD requirements.

Static SoD requirements, on the one hand, control the associations between entities during administration time. They would, for example, disallow a user to be assigned to a role if an SoD requirement would be violated. Dynamic SoD, on the other hand, does not restrict associations between entities at administration time. Instead, it controls the execution of permissions at run-time. It would, for example, allow a user to belong to the 'Manager' and 'Clerk' roles. However, during run-time, the user that initiated the purchase order (using the 'Clerk' role) will not be able to approve that purchase order (using the 'Manager' role).

The specification of both Static and Dynamic SoD requirements within the SoDA prototype is similar. This will be discussed in Section 4. Static SoD requirements must, however, also be enforced in the administration environment. The enforcement of Static SoD requirements in the SoDA prototype is thus discussed in Section 5.

4 Separation of duty specification in SoDA

The SoDA prototype allows for the specification of conflicting users, conflicting roles, conflicting permissions and conflicting tasks. A distinction is made between static and dynamic SoD. Conflicts are based on the sets U, R, P and T, representing the user, role, permission and task entities respectively. P is defined as $P \subseteq 2^{O \times M}$, where O represents the objects and M the methods that may be performed. Note that not all the methods may necessarily be defined on all objects. Thus, the set of permissions is a subset of the power set.

The specification of the conflicts is done through the sets:

$$CU_D, CU_S, CR_D, CR_S, CP_D, CP_S, CT_D, CT_S.$$

The same naming convention is followed. CX denotes conflicting entities of type X, and the subscript indicates whether the conflict must be checked statically (CX_S) or dynamically (CX_D). The "conflicting entities" relations are defined in a symmetric and non-reflexive fashion:

$$CX_Y \subseteq X \times X \text{ such that } \forall x_i \neq x_j$$
$$(x_i, x_j) \in CX_Y \iff (x_j, x_i) \in CX_Y$$

The specification for all 8 sets can be derived by replacing X with the appropriate entity (U,R,P or T) and Y with S or D, for Static and Dynamic respectively.

Figure 3 shows how conflicting tasks are identified within the SoDA prototype. The other conflicts are specified in a similar manner. The interpretation of the various conflicts is summarized in Table 1.

The enforcement of Dynamic SoD requires interpretation of the process instance. Thus it is the responsibility of the workflow system. Consequently, it falls outside the scope of the administrative tool. For a more detailed discussing regarding dynamic SoD the interested reader are refered to [3]. Static SoD must, however, be enforced in the administration environment. The next section discusses how this is implemented in the SoDA prototype.

Conflict	Static	Dynamic
Conflicting Roles	May not have the same user (or conflicting users) as members	May not be assumed by the same user (or conflicting users) in one process instance
Conflicting Permissions	Must be assigned to conflicting roles	May not be exercised by the same user (or conflicting users) for a specific process instance
Conflicting Users	May not belong to the same role or conflicting roles	May not perform conflicting tasks in the same process instance
Conflicting Tasks	Must be assigned to conflicting roles	May not be executed by the same user (or conflicting users) in the same process instance

Table 1: Interpretation of conflicts according to the "conflicting entities" paradigm

5 Static Separation of Duty enforcement in SoDA

In order to enforce Static SoD, the SoDA prototype ensures that the integrity of the associations between entities is maintained. If an action cannot be performed, remedial actions are suggested. For example, if conflicting tasks are assigned to non-conflicting roles, the user is given the option of making the roles conflicting. The associations that are allowed are summarized in Table 2 [9].

To illustrate how the SoDA prototype maintains the associations, this section will review different static SoD implementations of the requirement: "A person who issues stock may never approve an order". Three approaches to enforcing this SoD requirement in a static fashion are proposed. This is done by rephrasing the SoD requirement in the following ways:

(SoD1) A manager and a stock controller may not perform the same tasks.

(SoD2) The 'Issue Stock' permission and the 'Approve Order' permission may not be assigned to the same user.

(SoD3) The 'Issue Stock' task may not be performed by someone who performs the 'Approve Order' task.

These SoD constraints will be implemented as conflicting roles, conflicting permissions and conflicting tasks. Conflicting users can be used in combination with these.

Conflicting users are interpreted in the same way as in [AS99]. If two users are conflicting, it means that the chances of them colluding are very high. In essence, they should, therefore, be treated as if they were one user. For example, if two tasks may not be performed by the same user, two conflicting users may not perform them either as the chances of a conspiracy are high. We shall now consider how each of the approaches can, in turn, be handled in the prototype.

5.1 Conflicting Roles

First consider **(SoD1)** - A manager and a stock controller may not perform the same tasks.

Since managers approve orders, and stock controllers issue stock, the 'Manager' role in the 'Admin' role network and the 'Stock Controller' role in the 'Stores' role network may be set to conflict. Due to the inheritance property of role networks, conflicting roles cannot exist in the same role network. If conflicting roles were allowed in one role network, the topmost role in that role network would inherit the permissions of both conflicting roles. This clearly defeats the purpose. A role may conflict with more than one role in another network. Conflicts are, however, inherited up the partial order and setting more than one conflict, as such, may not be necessary. The SoDA prototype will remove any unnecessary conflict.

In Figure 4, the 'Stores Manager' inherits the conflict set upon 'Stock Controller'. 'Stores Manager' will, therefore, also conflict with the 'Manager' role in the 'Admin' role network. In Figure 3, the 'Approve order' and 'Issue stock' tasks were made conflicting tasks. Conflicting roles and conflicting tasks impact on the allowable associations as follows. Only non-conflicting users may be assigned to conflicting roles. Conflicting tasks must be performed by conflicting roles. Recall that the 'Stock Controller' role and the 'Stores Manager' role were identified as conflicting with the 'Manager' role. Figure 5 depicts the 'Manager' role as being assigned to the 'Approve Order' task. Figure 5 shows, furthermore, that subsequently only the two roles conflicting with the 'Manager' role, namely the 'Stock Controller' and 'Stores Manager' roles, may be assigned to the 'issue stock' task. If two tasks are initially not indicated to be conflicting, but they are assigned to conflicting roles, the tasks are made conflicting tasks.

5.2 Conflicting Permissions

Now consider **(SoD2)** – The 'Issue Stock' permission and the 'Approve Order' permission may not be assigned to the same user.

The permissions involved are editing the 'Approval'

May be associated with		Roles	
		Conflicting	Non-conflicting
Users	Conflicting	N	Y
	Non-conflicting	Y	Y
Permissions	Conflicting	Y	N
	Non-conflicting	Y	Y
Tasks	Conflicting	Y	N
	Non-conflicting	Y	Y

Table 2: Static SoD – Allowable associations

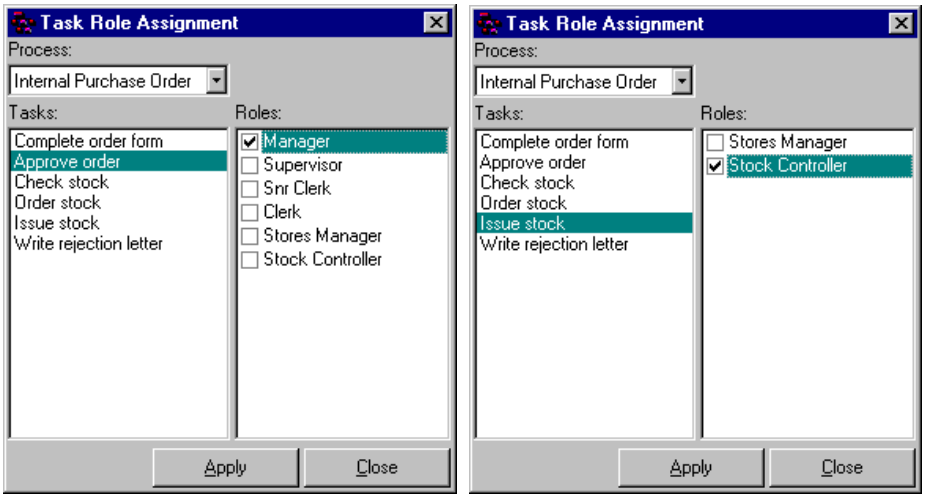

Figure 5: Conflicting tasks must be assigned to conflicting roles

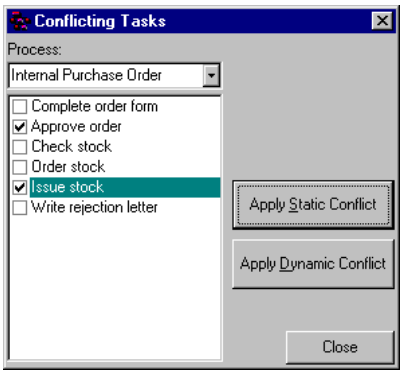

Figure 3: Specifying conflicting tasks

and 'IssueRec' field groups on the 'Internal Order Form' object. Conflicting permissions may only be assigned to conflicting roles. If this is not enforced, conflicting permissions could be assigned to conflicting users. These conflicting users belong to non-conflicting roles, which have conflicting permissions that were incorrectly assigned to the non-conflicting roles. This clearly opens the door for a conspiracy. The SoDA prototype, therefore, only allows conflicting roles to receive conflicting permissions.

If the roles are not conflicting, they are made conflicting, subject to additional integrity checking. Roles cannot be made conflicting if conflicting users are assigned to the said roles. It can, therefore, be seen that even if the 'Man-

Figure 4: Conflicting roles

ager' and 'Stock Controller' roles were not initially identified to be conflicting, they will be made conflicting when the two conflicting permissions are assigned to these two roles. Similar to section 4, the tasks assigned to these two roles will also be made conflicting.

5.3 Conflicting Tasks

Consider **(SoD2c)** – The 'Issue stock' task may not be performed by someone who may perform the 'Approve order' task. In section 5.1, it was shown how conflicting roles could only be assigned to conflicting tasks. If conflicting roles were assigned to tasks, these tasks were automatically made conflicting. This approach can be considered to be the reverse of that. Two tasks are defined to be conflicting. Subsequently, the roles that must be assigned to the user must be conflicting. If two non-conflicting roles are assigned, the roles are made conflicting, subject to a series of integrity checks being performed. It is evident that the same result is achieved, irrespective of the approach used, since automatic maintenance of conflict relationships is performed.

The results of the conflicting role and conflicting task approaches are thus identical. The conflicting permission approach can, however, be considered stricter. Conflicting permissions must be performed by conflicting roles. However, conflicting roles do not only have conflicting permissions. For example, the 'Manager' and 'Stock Controller' roles are conflicting, but both should still be allowed the 'view purchase order' permission. The conflicting permissions 'Edit Approval' and 'Edit Issuerec' may, however, also be assigned to the 'Manager' and 'Stock Controller' roles respectively.

6 Conclusion

This paper demonstrated the "conflicting entities" paradigm as a way of specifying SoD requirements. This paradigm uses the task abstraction to intuitively define separation of duty requirements that involve sequence of operations. It was shown that both Static and Dynamic SoD requirements can be formulated according to the "conflicting entities" paradigm in the SoDA prototype.

It was, furthermore, shown that the SoDA prototype enforces Static SoD requirements. By specifying one SoD requirement in three different ways, it was explained that equivalent results can be achieved.

It should be noted that Static SoD requirements are extremely restrictive on the organizations functioning. Consider, for example **(SoD1)**. To assume that a managers and a stock controller could never do the same job could be, especially for a small company, very restrictive. Dynamic SoD requirements addresses this issue by imposing the restrictions per process instance.

Other issues that could be of concern are the potential of a lock-out situation. A situation could arise that, for example no roles are available to assign to a task. This would immediately be noticable to the system administrator and he/she will have to rectify the situation manually. However, due the extremely strict restrictions imposed by static separation of duty, it is likely to be used sparingly. This makes the likelihood of a lock-out occurring extremely small and thus feasible for the adminstrator to manually correct. The issue of lock-out occuring due to dynamic SoD requirements are much more complex and state-of-the-art work regrading that may be found in [2].

References

[1] G-J. Ahn and R. S. Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th ACM Workshop on Role-based Access Control*, pages 43 – 54, 28 – 29 Oct. 1999.

[2] E. Bertino, E. Ferrari, and V. Atluri. Specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security*, 2(1):65–104, Feb 1999.

[3] R. A. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3), 2001.

[4] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184 – 194, Apr. 1987.

[5] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transaction on Information and System Security*, 2(1):34 – 64, Feb. 1999.

[6] D. Hollingsworth. The workflow reference model. Technical Report TC-00-1003, Workflow Management Coalition, www.wfmc.org, Jan 1995.

[7] D. R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of the 2nd ACM Workshop on Role-based Access Control*, pages 23 – 30, Oct. 1997.

[8] M. Nyanchama and S. Osborn. The role-graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3 – 33, Feb. 1999.

[9] S. Perelson and R. A. Botha. Conflict analysis as a means of enforcing static separation of duty requirements in workflow environments. *South African Computer Journal*, (26):212 – 216, Nov. 2000.

[10] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of IEEE*, 63(9):1278 – 1308, 1975.

[11] R. S. Sandhu, E. J. Coyne, H. L. Fenstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38 – 47, Feb 1996.