Separation of duties for access control enforcement in workflow environments

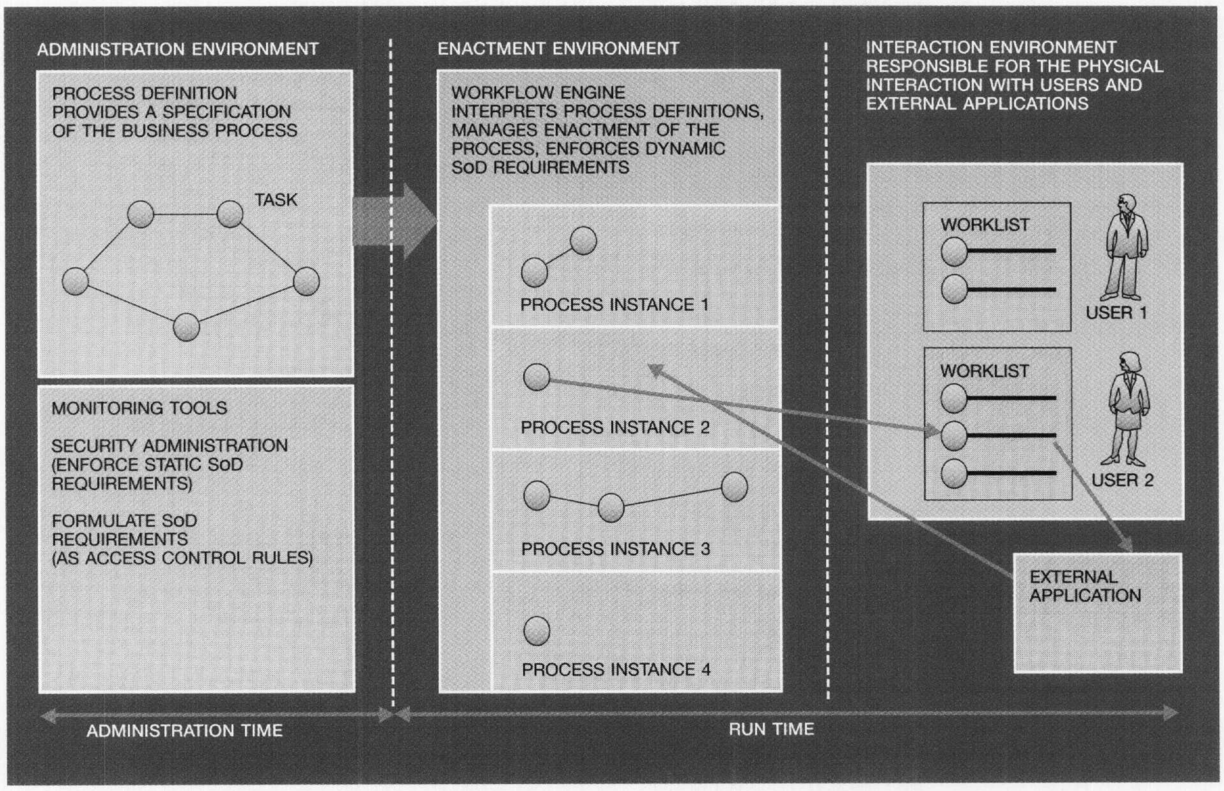
by R. A. Botha
J. H. P. Eloff

Separation of duty, as a security principle, has as its primary objective the prevention of fraud and errors. This objective is achieved by disseminating the tasks and associated privileges for a specific business process among multiple users. This principle is demonstrated in the traditional example of separation of duty found in the requirement of two signatures on a check. Previous work on separation of duty requirements often explored implementations based on role-based access control (RBAC) principles. These implementations are concerned with constraining the associations between RBAC components, namely users, roles, and permissions. Enforcement of the separation of duty requirements, although an integrity requirement, thus relies on an access control service that is sensitive to the separation of duty requirements. A distinction between separation of duty requirements that can be enforced in administrative environments, namely static separation of duty, and requirements that can only be enforced in a run-time environment, namely dynamic separation of duty, is required. It is argued that RBAC does not support the complex work processes often associated with separation of duty requirements, particularly with dynamic separation of duty. The workflow environment, being primarily concerned with the facilitation of complex work processes, provides a context in which the specification of separation of duty requirements can be studied. This paper presents the "conflicting entities" administration paradigm for the specification of static and dynamic separation of duty requirements in the workflow environment.

The proliferation of computer networks facilitated the move toward office automation. Allen[1] identifies the move toward automating the office as one of the turning points in interacting with computers in the previous millennium. The convergence of computing, communications, and digital information has enabled business activities to be supported across boundaries previously deemed unsurpassable. Workflow, being the computerized facilitation of business processes,[2] had become a much-discussed topic in the 1990s. Workflow management systems provide the facilities to define, manage, and execute business processes in an electronic fashion. Increasing use of electronic means to conduct business leads to significant increases in processing performance and efficiency. These advantages, however, come at a cost. One such cost is an increased information security risk.

Information security requires systems to provide five essential services, namely authentication, access control, integrity, confidentiality, and nonrepudiation.[3] These security services protect various attributes. Take, for example, the integrity attribute. This attribute is concerned with three different aspects of integrity:[4] (1) operational integrity deals with concurrent access to data, (2) physical integrity implies protection from loss of data, and (3) semantic integrity requires that the data comply with the appropriate business rules. Operational integrity can be maintained through concurrency controls as part of the integrity service. Physical integrity can be achieved by the use of checksums, provided by the integrity service, and cryptographic techniques, pro-

Figure 1 The workflow environment


ADMINISTRATION ENVIRONMENT
PROCESS DEFINITION PROVIDES A SPECIFICATION OF THE BUSINESS PROCESS
TASK
MONITORING TOOLS
SECURITY ADMINISTRATION (ENFORCE STATIC SoD REQUIREMENTS)
FORMULATE SoD REQUIREMENTS (AS ACCESS CONTROL RULES)
ENACTMENT ENVIRONMENT
WORKFLOW ENGINE INTERPRETS PROCESS DEFINITIONS, MANAGES ENACTMENT OF THE PROCESS, ENFORCES DYNAMIC SoD REQUIREMENTS
PROCESS INSTANCE 1
PROCESS INSTANCE 2
PROCESS INSTANCE 3
PROCESS INSTANCE 4
INTERACTION ENVIRONMENT RESPONSIBLE FOR THE PHYSICAL INTERACTION WITH USERS AND EXTERNAL APPLICATIONS
WORKLIST
USER 1
WORKLIST
USER 2
EXTERNAL APPLICATION
ADMINISTRATION TIME
RUN TIME

vided by the confidentiality service. Semantic integrity relies heavily on the access control service, because it requires data to be changed only according to certain business rules.

An example of such business rules is separation of duty requirements. Separation of duty requirements have as their primary objective the prevention of fraud and errors, thus ensuring semantic integrity. They are often formulated as business rules such as "a check requires two different signatures." This rule needs the cooperation of the authentication service to confirm the identity of the user and the access control service to limit access rights at particular times.

This paper deals with the implementation of separation of duty requirements in the access control service in a workflow environment. In the ensuing presentation, role-based workflow environments are introduced first. Subsequently, the "conflicting entities" administration paradigm is introduced. Thereafter it is demonstrated how this paradigm applies to static separation of duty requirements. Finally, dynamic separation of duty requirements are demonstrated according to the "conflicting entities" administration paradigm.

We start by observing a typical role-based workflow environment.

Role-based workflow environments

Figure 1 depicts the three main functional areas that can be observed in a workflow system, namely the administration environment, the enactment environment, and the interaction environment. These areas are briefly considered in turn.

The *administration environment* is used to define the business process by means of a process definition. The basic building blocks of a process definition are tasks.[5] Tasks represent a small unit of work. Human users could perform tasks, or tasks could be automated to represent computer activity such as the updating of a database. Tasks to be done by humans are the responsibility of a single user. The computer-understandable expression of a task is called the task definition.

A business process is defined by linking tasks in a directional graph with predicate conditions on the arcs of the graph. The computer-understandable form of a business process is called the process definition. When the business process is performed, the predicates will be evaluated on completion of a task to determine which task(s) must be performed next.

In addition to the definition of business processes, the administration environment will also be used to monitor workflow enactment, manually intervene in automated activities, and perform security administration tasks.

Separation of duty (SoD) requirements will be formulated in the administration environment as business rules. As an example, a business rule may state: "A person may not approve his or her own purchase order."

In the *enactment environment* the workflow engine is responsible for interpreting the business process definitions and creating a process instance for each time the process must be enacted. A process instance can thus be seen as a persistent object representing a specific occurrence of a business process. Each process instance will record information pertaining to that specific process instance. Examples of such information are: the initiator of the process, the date or time of initiation, the tasks that have been performed, and any other information that has an influence on the execution of the business process. The enactment environment considers the state of the process instance, evaluates the predicates that determine the route to follow, and creates the appropriate task instance. Each process instance thus consists of one or more task instances.

Furthermore, the enactment environment will determine which users may perform the task. In order to make this decision, the enactment environment evaluates the separation of duty requirements that were defined as part of the access control rules in the administration environment. It would thus decide not only *what* the next task is but also *which* users may possibly perform that task. In the case of a purchase order it would, for example, determine that the "approve order" task is the one to be performed. In addition, it would exclude the initiator of the purchase order from the potential performers of the activity.

The *interaction environment* is concerned with the actual performance of the task by the user. A common implementation would use the task list metaphor, also called a worklist, to present the user with the tasks that he or she must (or can) perform. Although several implementations of a worklist are possible, the model suggested in this paper requires the worklist to be dynamically constructed during run time. Each time that a user attempts to perform a task on the user's worklist, the interaction environment must confirm that the user is still authorized to perform the task.

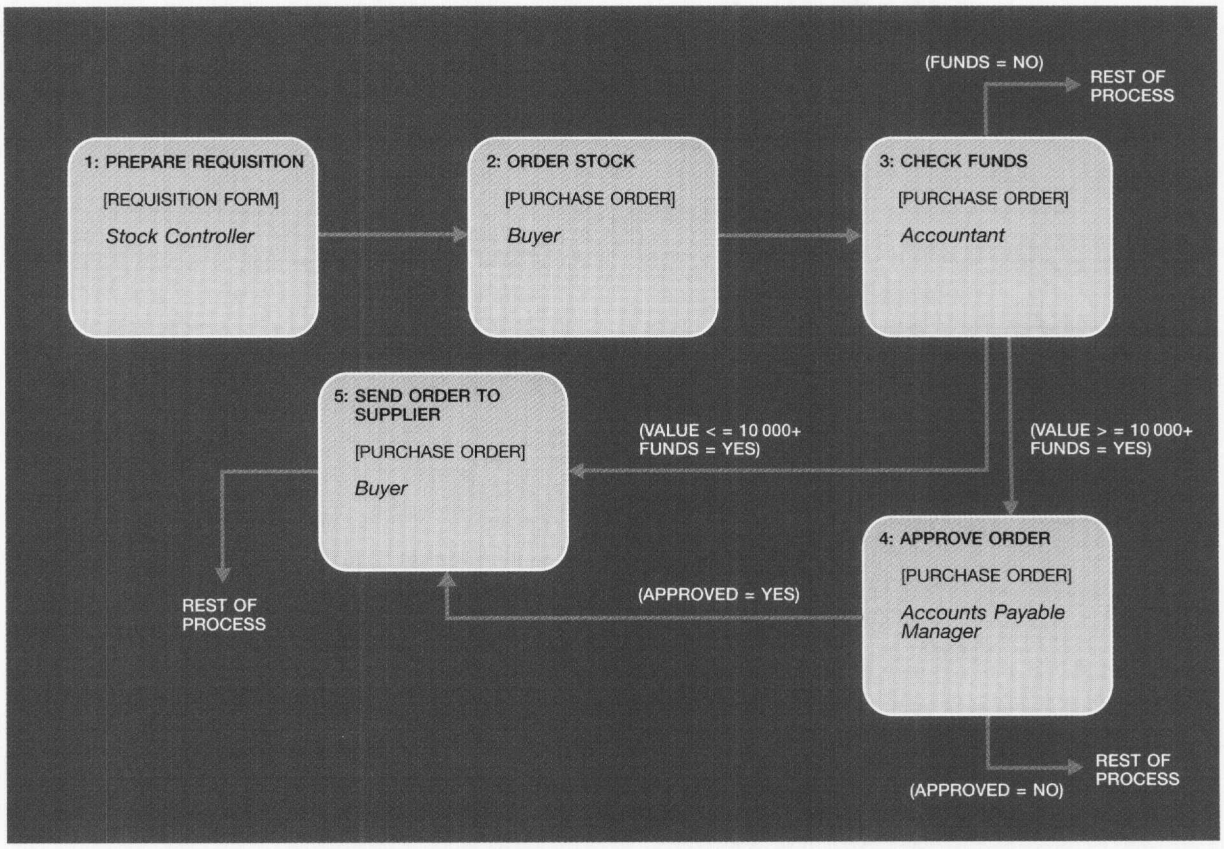
The interaction environment may activate other applications based on the requirements of a task. A word processor, for example, may be activated when the task requires a letter to be written. It must further ensure that the access rights are granted and revoked in a timely fashion. For this reason, it is responsible for ensuring that access rights are only available to a user while actually performing a task. Access rights can, therefore, not be abused outside the task context. It would, for example, ensure that the "approve order" authorization, or permission, is not available to a person when the purchase order is initiated.

In order to discuss the concepts further, we introduce an example of a workflow that will be used throughout the paper for illustrative purposes.

A workflow example. Figure 2 graphically describes a portion of a typical purchasing workflow. Each task on the diagram is divided into three sections: (1) the description of the task is identified in bold type, (2) the document being used is identified between square brackets, and (3) the organizational position of the person performing the task is identified in italics. On completion of a task, the workflow engine determines the next task to be initiated by evaluating the predicates shown between parentheses on the line connecting tasks.

In the portrayed workflow, a buyer will complete a purchase order (Task 2) based on a requisition form prepared by the stock controller (Task 1). Thereafter an accountant will determine the availability of funds (Task 3). Should funds be available, the next step is determined by the value of the order. Orders in excess of 10 000 are forwarded to the accounts payable

Figure 2 A purchase process

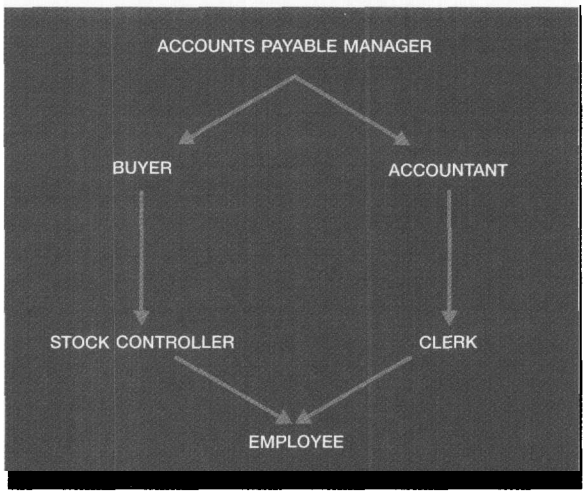


manager for approval (Task 4), whereas smaller orders are immediately sent to the supplier by the buyer (Task 5). Although the business process consists of several more tasks linked by several possible routes, the other tasks bear no impact on the rest of the paper and are thus omitted.

The workflow example utilized documents as the primary interface to the various tasks. These documents could contain sensitive information. Access to this information thus needs to be tightly controlled. The example furthermore showed that tasks are often associated with organizational positions, also called roles. Role-based access control (RBAC) has become a popular mechanism for administering access control, both in and out of workflow environments.[6-9] RBAC also plays a fundamental role in current separation of duty research. It is therefore appropriate to discuss role-based access control, in particular as it pertains to the workflow environment. Thereafter a state-of-the-art review of research regarding separation of duty is given.

Role-based access control. RBAC uses the role abstraction as its primary means of specifying access control requirements.[10] During *role administration*, permissions are assigned to roles and users are assigned to roles. Permissions indicate the ability to perform a certain operation on information. "Create order" and "approve order" may, for example, be two permissions in the current example. A user is assigned membership in a role consistent with the user's duties and responsibilities in the organization. Role administration is reduced by the incorporation of a role hierarchy. A role hierarchy indicates a seniority relationship between roles, typically according to the management structure of the organization. Figure 3 depicts a role hierarchy relating to the example workflow. Permissions are inherited upwards in the hierarchy. Thus, any permissions assigned to

Figure 3 Role hierarchy example



the "stock controller" role are inherited by the "buyer" role.

Users, however, do not always use or need all of the permissions specified during role administration. Role-based environments thus provide a means to control *role activation*. Control is achieved through the notion of a session. By definition, a session associates users at run time with a subset of the roles that may be assumed by the user. This association may be necessary, for example, when Tom, an accounts payable manager, is acting as a buyer and creates a purchase order. At that stage, Tom should not have permission to "approve order."

RBAC is strongly featured in current research on separation of duty. The next subsection provides an overview of state-of-the-art research regarding separation of duty.

Separation of duty. Saltzer and Schroeder[11] identified separation of duty, 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[12] identified separation of duty as a mechanism that can be implemented to ensure data integrity by making certain that system objects correspond to the real-world objects they represent. They asserted that, at the policy level, processes are divided into steps, with each step performed by a different person. Separation of duty is thus tightly connected to application semantics.

Sandhu's work on Transaction Control Expressions[13,14] introduced a notation for dynamic separation of duty. Roles were used to specify which users could perform which transaction steps. Separation of duty was then introduced by specifying that a user may only perform one step in a transaction. In order to achieve this one-for-one relationship, a history of the execution of each step had to be kept with the object on which the transaction operated. A study by Nash and Poland[15] introduced the phrase object-based separation of duty, meaning that every transaction against an object is forced to be performed by a different user. Sandhu et al.[10] presented a formal model of RBAC with the concept of constraints on the associations firmly embedded.

The paper by Ferraiolo, Cugini, and Kuhn on RBAC[16] presented operational separation of duty as a supplement to static and dynamic separation of duty. Operational separation of duty required that no role could contain the permissions for all the operations necessary to perform a business process. This requirement implies that at least two roles should be involved in the completion of each business process.

Simon and Zurko[17] enumerated various forms of separation of duty and indicated how the separation of duty policies can be expressed in Adage, a general-purpose access control policy editor.

Kuhn[18] explored the mutual exclusion of roles as a separation of duty mechanism. In addition to the time at which mutual exclusion is applied (static versus dynamic), the degree to which privileges are shared by mutually exclusive roles (strong or partial exclusion) are also considered.

Gligor, Gavrila, and Ferraiolo[19] also enumerated the known forms of separation of duty requirements by presenting a formal model to express separation of duty constraints in an RBAC environment. They indicated the relationships between the various separation of duty constraints. Their work, however, failed to address role hierarchies in typical RBAC implementations.

Nyanchama and Osborn[20] discussed various kinds of conflicts that have to be considered when implementing separation of duty policies. Their model does not involve separation of duty principles that require a model of task dependencies. They evaluated the effect of role hierarchies in great depth in terms of their role-graph model.

Ahn and Sandhu[21] defined the RSL99 language for specifying separation of duty constraints. They based their separation of duty requirements on the concepts of conflicting entities. Sets of conflicting users, conflicting roles, and conflicting permissions restrict the way in which users, roles, and permissions can be associated with one another.

None of the mentioned papers addresses the notion of a task, although some do recognize this omission as a shortcoming of their models.

Thomas and Sandhu[22-24] proposed that access control modeling should be approached from the perspective of activities or tasks, thus yielding an "active" security model. The model is active in the sense that it provides mechanisms and abstractions for the run-time management of access control permissions as tasks progress to completion. Their papers provide excellent motivation for the necessity of controlling access dynamically. They establish a new conceptual framework for their active security model. It is not shown, however, how this task-based model integrates with existing access control models. This integration is a necessary requirement within an enterprise environment, because not all access control decisions in an organization will be made in the context of a workflow.[25]

The model proposed by Bertino et al.[6] presents state-of-the-art thinking regarding access control in the workflow environment. Their model addresses the expression and evaluation of access control constraints in great detail with a formal language. The expression of constraints in a formal language is, however, unsuitable for end-user specification. The work presented in this paper concentrates on employing a paradigm for the administration of separation of duty requirements. This is done in an attempt to minimize the impedance mismatch between the administration and enforcement paradigm.

Even a cursory glance at the preceding material makes it clear that the workflow environment requires special access control functionality with special administrative needs. The next section thus introduces the "conflicting entities" administration paradigm as a unified way to administer separation of duty requirements within a workflow environment.

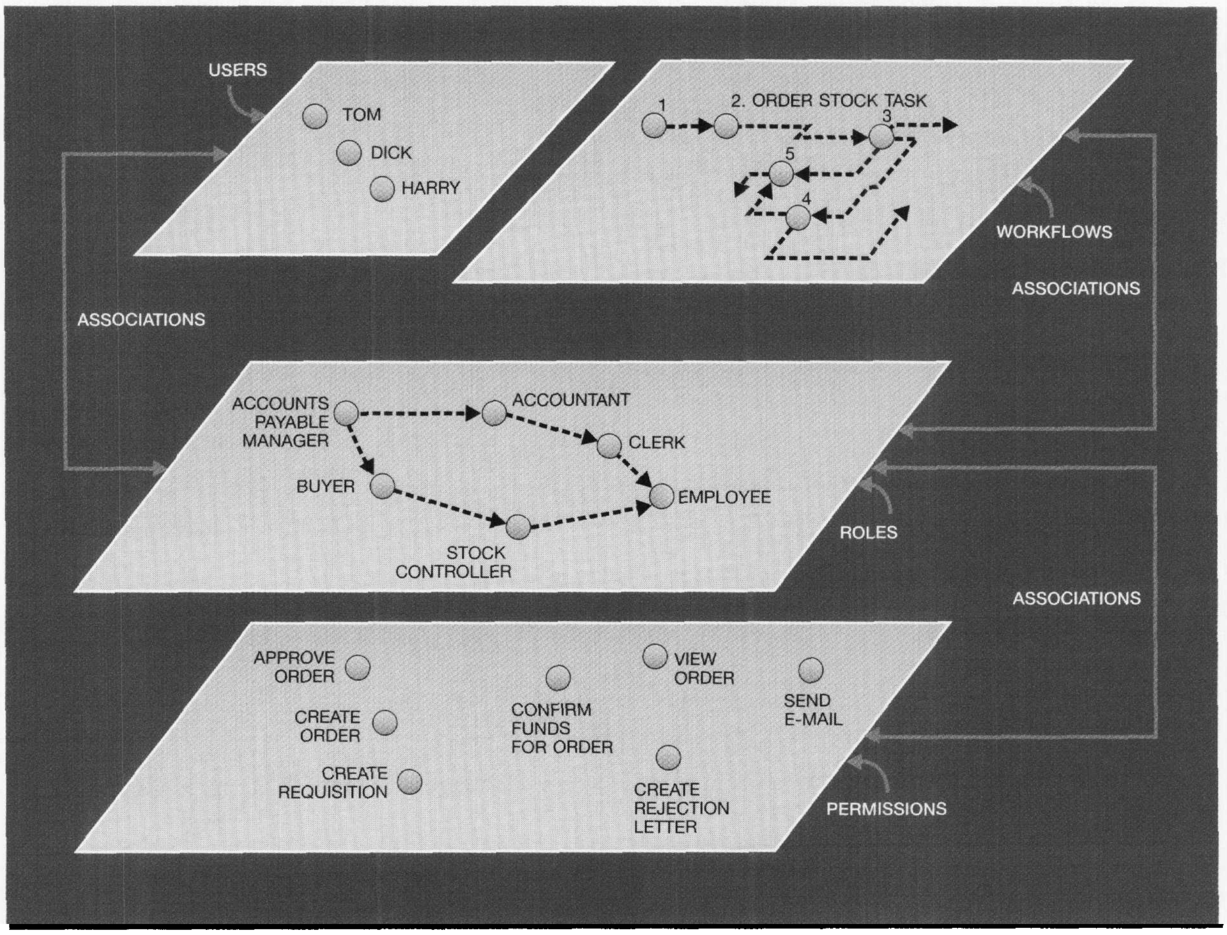
CoAP: The "conflicting entities" administration paradigm

Many of the current models contain the notion of "conflict" in some guise or other. However, none of the work associates conflict with all the entities that could conflict at the same time. Notable, in most of the cases, is the lack of reference to a task abstraction. Exceptions are the work of Thomas and Sandhu[22-24] and the work of Bertino et al.[6] In Reference 23 conflicts between tasks (called separation-dependency) are identified as an important way of expressing separation of duty. However, the influence and existence of other potential conflicts are not discussed. In Bertino et al.,[6] conflicts between tasks are represented through a number of predicates that form part of their formal language. However, it is considerably more difficult to express separation of duty requirements such as, "Since John and Jack are brothers, they should not be able to approve each other's orders," than, "Since John initiated the order he may not approve it."

This paper subsequently presents the "conflicting entities" administration paradigm (CoAP) as a uniform way to specify separation of duty constraints in the workflow environment. In this way, the various entities that have been identified in previous research, namely users, permissions, roles, and tasks, are included. This paper recognizes that each of those entities could have conflicting elements in them. Conflict between entities, in the general sense, implies that the risk of fraud increases if associations with those entities are not carefully controlled. Figure 4 depicts the entities and associations evident in CoAP. The task entity is upheld as the building block for expressing separation of duty requirements with CoAP in terms of the underlying workflow models. Before presenting a detailed discussion of the conflicting entities, we consider a brief, high-level discussion of the four types of conflict that can be identified.

Fundamental to the interpretation of "risk of fraud" is the concept of permissions. Permissions indicate the ability to perform a certain action on an object. Permissions, therefore, are central to our separation of duty definitions. Permissions will be considered as *conflicting permissions*[20] if they, together, provide more ability than required to a single user. *Conflicting users*[21] are those users who may stand to gain by conspiring. In practice such users may be family

Figure 4 Entities and associations in CoAP



members. *Conflicting roles*[18,20] are positions that share conflicting permissions. From a practical perspective, the conflicting permissions might not always be identified, since the identification of conflicting permissions for conflicting roles may negate the administrative advantages obtained through the role abstraction in RBAC. However, identifying conflicting roles would be senseless if there were no conflicting permissions involved. *Conflicting tasks* similarly require some conflicting permissions to complete. To ease administration, conflicting tasks may also not enforce the identification of conflicting permissions.

These conflicts may be checked either at administration time or at run time (see Figure 1). The next subsection discusses how these conflicts can be evaluated at administration time, i.e., to enforce static separation of duty requirements. Thereafter, how dynamic separation of duty requirements can be expressed with CoAP is discussed.

Static separation of duty requirements. Static separation of duty requirements can be enforced in the administration environment. For this reason, assignments that should never occur can thus be prohibited. It has been pointed out that this feature can be very restrictive to business operations, especially in smaller organizations. However, it may still prove useful in cases where business rules that span an entire organization and stay constant over time must be expressed.

Consider the well-accepted business rule that "auditors should act independently." This rule implies

Table 1 Static separation of duty interpretations for the business rule "auditors should act independently"

Possible Conflict	Interpretation with Reference to Business Rule
Conflicting roles	User assignments to the "auditor" role and (for example) "accounts payable manager" role must be mutually exclusive.
Conflicting permissions	The same user may under no circumstances receive the "approve order" and "approve audit" permission.
Conflicting users	Members of the same family must be considered as the same user and may therefore not be assigned to roles, permissions, or tasks to which a single user should not be assigned.
Conflicting tasks	A user who can do the "approve order" task may never do the "approve audit" task, and vice versa.

that auditors should not be able to audit their own work. There are several interpretations of this business rule. Table 1 summarizes an interpretation in terms of each of the conflicting entities.

The first interpretation relies on the specification of *conflicting roles*. By making the role of auditor conflict with roles of authority, such as the accounts payable manager in the workflow example, the same user may never be assigned the role of auditor and the role of accounts payable manager.

The second interpretation utilizes the specification of *conflicting permissions*. It requires that permissions that could cause fraudulent audits to occur be identified as conflicting permissions. In practice this interpretation could prove very laborious and thus impractical. It, however, does have the advantage that an auditor could be allowed to perform some of the actions of a manager, provided that those actions would not likely lead to defrauding the audit process.

In the third interpretation the notion of *conflicting users* is incorporated. An auditor would potentially be biased should he or she audit the work of a family member. This condition could be specified by identifying family members as conflicting users. These conflicting users will be interpreted as a single user. For this reason, the conflicting user restriction would operate in conjunction with other conflicts. It would prohibit, for example, family members from membership in the auditor and accounts payable manager roles as specified in the first interpretation. The inflexibility of static separation of duty is also shown by this requirement whereby a person would not be able to ever act in the role of auditor if a family member of his or hers is in a role of authority such as the accounts payable manager.

Finally, *conflicting tasks* could be used to specify static separation of duty requirements. In this instance the auditing tasks could be identified as conflicting with tasks that have financial significance in the example workflow. For example, the "approve order" task in the example workflow would be conflicting with tasks in the audit orders process. This conflict would imply that roles (and therefore users) that can partake in the tasks in the audit orders process, for example, the "approve audit" task, would not be able to partake in the "approve order" task.

The formal relationships between statically conflicting entities were presented in the paper by Perelson and Botha.[26] It indicates that specifying static separation of duty requirements is indeed very restrictive. However, in certain cases the restriction may prove practical. For example, the requirement that auditors may not do anything but audit and that they should have no family ties that could make them even slightly biased might be realistic. The extreme restrictions imposed by static separation of duty can be alleviated by using dynamic separation of duty requirements, as explained in the next subsection.

Dynamic separation of duty requirements. Dynamic separation of duty requirements do not restrict associations in the administration environment. They, however, restrict the activation of those roles at run time according to the separation of duty specification. In RBAC,[10] the session concept is used to limit the run-time associations between roles and users. In the workflow environment, we introduce the concept of a WSession as a specialization of the session concept found in RBAC models.

WSessions in a role-based workflow environment. The workflow enactment environment strictly controls the establishment of WSessions. A WSession controls the activation of roles in the workflow environment based on the task instance with which the user currently deals. A WSession is transient in the sense that it will exist only while the user is busy with a particular task. Synchronization models such as that proposed by Atluri and Huang[27] could be implemented to ensure the timely creation and destruction of WSessions.

A WSession is a run-time mapping of one user to the most junior role required for a specific task.

The need for strict least privilege[28] in a workflow environment requires the user to assume the absolute minimum role required for the task. A distinction is thus made between the user's job and the tasks that he or she performs as part of the job. Strict least privilege therefore requires the permissions to change, depending on the specific task with which the user is busy. In order to support the concept of strict least privilege, a WSession is thus further constrained to only allow the activation of a single role. This role corresponds to the role associated with the task, which represents the minimum permissions required for that task. The user must be a member of the role or of a role senior to the role.

In summary, a WSession is a run-time *mapping* of one user to the most junior role required for a specific task.

The WSession concept allows Tom, an accounts payable manager, to assume the buyer role without assuming the accounts payable manager role. Figure 5 shows how a WSession is a pivotal element in linking users, roles, and the workflow definition. Tom, who acted on a worklist item that only requires the permissions of the buyer role, receives only the permissions associated (directly or through inheritance) with the buyer role. Tom's administration-time associations are indicated by the white lines in Figure 5. Tom, in this case, is only granted the "create order," "create requisition," "view order," and "send e-mail" permissions as indicated by the red lines in Figure 5. These lines represent the run-time associations based on the buyer role required by the active "order stock" task.

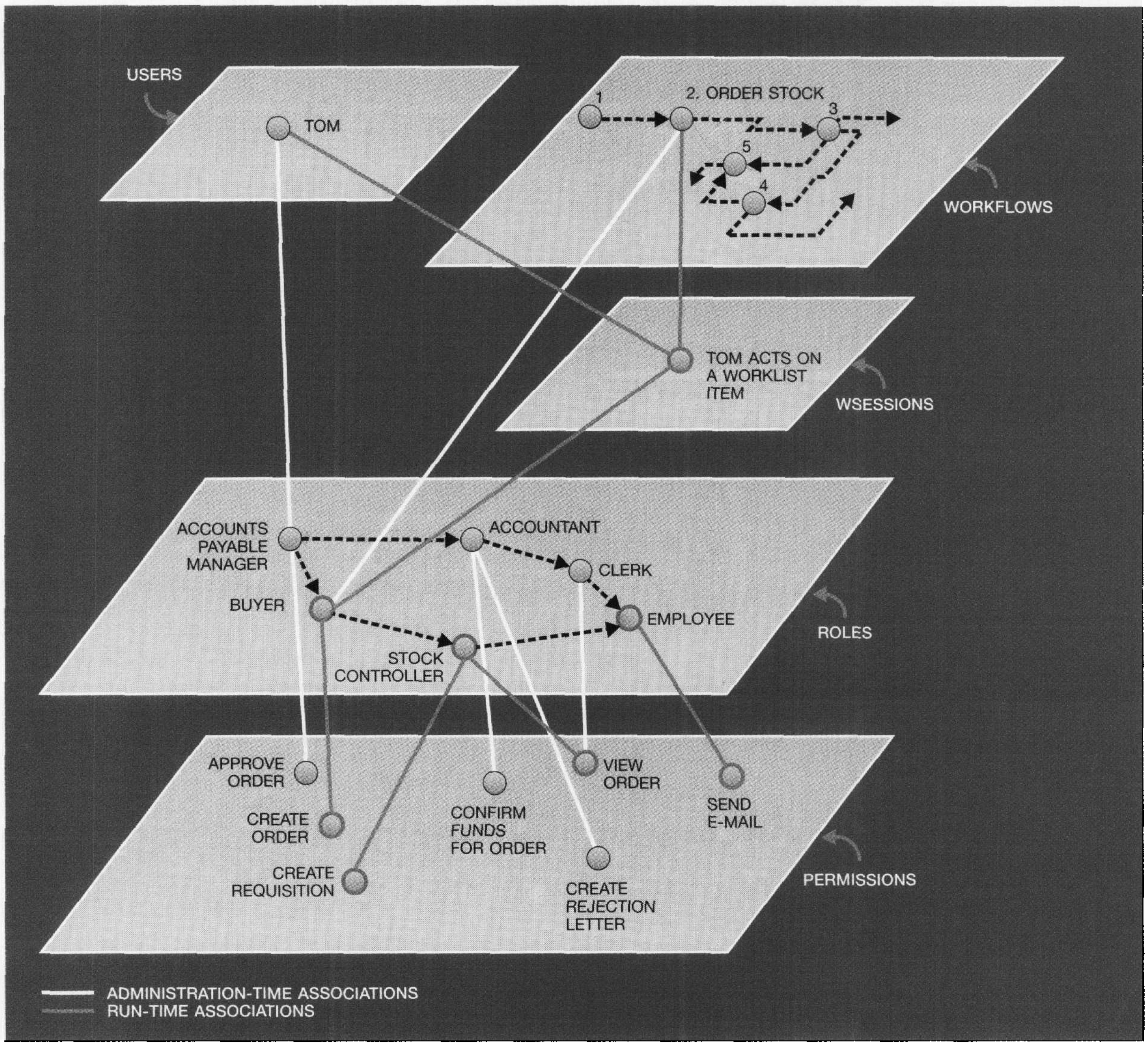
The concept of a WSession thus refers to the time from when a user starts to work on a task in the worklist to when that user stops or suspends work on the same task. A task instance may thus require several WSessions to be completed. The workflow engine is responsible for keeping track of the state of a task instance. Once a user selects an item from the worklist, the user takes responsibility for the corresponding task instance. The user must therefore complete that task instance. The user that accepted responsibility for a task instance will thus be described by an attribute of that task instance. As soon as the user suspends or completes the task, all permissions are revoked from the user, and the WSession is destroyed.

Now we consider how the WSession concept can be used to enforce dynamic separation of duty requirements specified according to the "conflicting entities" administration paradigm.

The implementation of dynamic separation of duty with the use of WSessions. Dynamic separation of duty requirements do not restrict associations in the administration environment. They, however, restrict the activation of those roles in the sessions according to the separation of duty specification. In the workflow environment, the dynamic separation of duty requirements are analyzed when the worklists are generated. Only users who may perform the task will be notified in their worklists. Once a user selects a task from the worklist, the separation of duty requirements and the user's role are again evaluated to ensure that only legitimate users perform the task. This check is necessary since a user's permissions might have changed since the creation of the worklist. Table 2 summarizes how the four types of conflict can be used to interpret the business rule: "An order may not be approved by its initiator."

Dynamically conflicting roles are presented as the first possible approach. If the stock controller and the accounts payable manager roles are identified as dynamically conflicting roles, they may not be activated for the same person in one process instance. It is important to recognize that the roles of stock controller and accounts payable manager in the example workflow are related through a role hierarchy. If a user therefore assumes the accounts payable manager role, the stock controller role is inherited by him or her. This inheritance would give

Figure 5 The WSession concept in a workflow environment



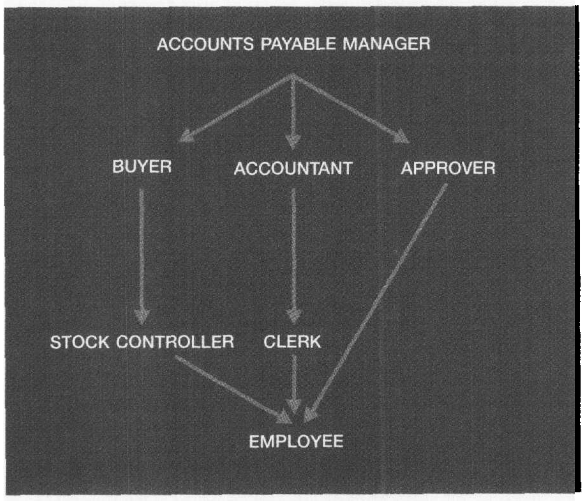
unnecessary permissions to a manager who wishes to approve an order, thus violating the strict least privilege requirement. To allow use of the dynamically conflicting role approach, the role hierarchy needs to be re-engineered as in Figure 6. The "approve order" task that is associated with the accounts payable manager role must therefore be associated with the new "approver" role depicted in Figure 6.

Dynamically conflicting permissions may be identified as the second approach. In this instance the "approve order" and "create requisition" permissions may be identified as dynamically conflicting permissions. If, for example, Tom has exercised the "create requisition" permission in a specific process instance, he will not be able to assume the accounts payable manager role since doing so will allow him to exercise the "approve order" permission. For another process instance, however, Tom will be allowed to obtain the permission "approve order" if he did not exercise a "create requisition" for that specific process instance.

Table 2 Dynamic separation of duty interpretation of the business rule "an order may not be approved by its initiator"

Possible Conflict	Interpretation with Reference to Business Rule
Conflicting roles	The "stock controller" role and the "accounts payable manager" role may not both be activated in a single user's WSessions for one process instance.
Conflicting permissions	The "create order" and "approve order" permissions must not be exercised in WSessions of the same user for a process instance.
Conflicting users	Family members (conflicting users) must not operate on conflicting tasks or exercise conflicting permissions in any WSessions of a process instance.
Conflicting tasks	The "create requisition form" task and the "approve order" task must be done in WSessions belonging to different people for each process instance.

Figure 6 Role hierarchy adapted to allow for strict least privilege



The third interpretation relies on the specification of *dynamically conflicting users*. If, for example, Tom and Dick are considered likely to conspire, they may be identified as conflicting users. Tom and Dick would thus be considered as one user in the context of each process instance. The implications are considered with other restrictions imposed on a single user, such as conflicting permissions. If dynamically conflicting permissions are as above and Tom and Dick are conflicting users, it would imply that if Tom exercised the "create requisition" permission, it is seen as if Dick did it as well. Dick would therefore also not be allowed to activate a role that would provide him with the "approve order" permission for that process instance. In another process instance where neither Tom nor Dick exercised the "create requisition" permission, either Tom or Dick would be able to assume a role granting him the "approve order" permission.

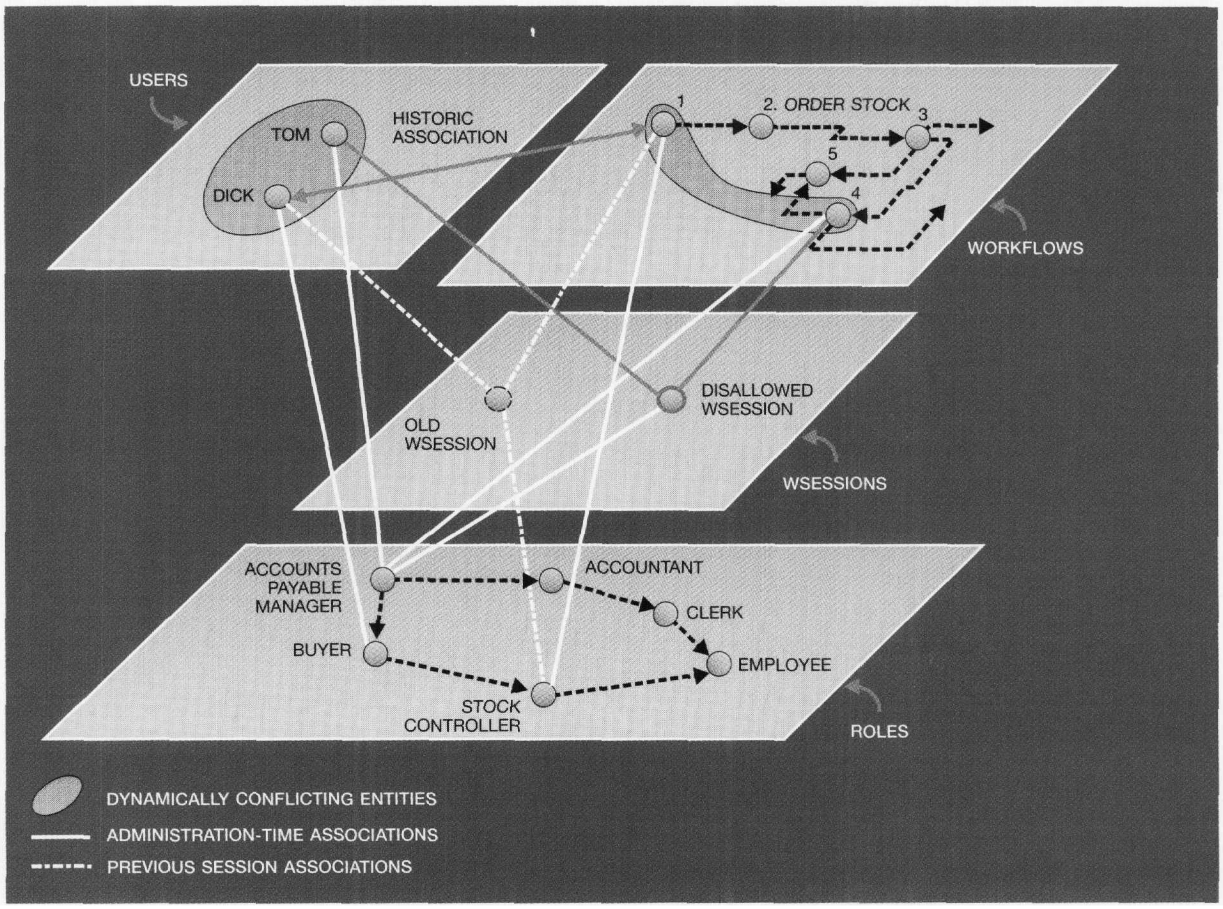
Finally, *dynamically conflicting tasks* provide an alternative way to specify the business rule. In the current example, it may be specified that the "create requisition form" task and the "approve order" task are dynamically conflicting. If a user, say Tom, did the "create requisition form" task, he will not be allowed to act in the "approve order" task.

Figure 7 is a graphical depiction of how the business rule, "an order may not be approved by its initiator," is enforced using conflicting tasks and conflicting users. In a previous WSession, Dick acted as the stock controller that performed Task 1, "create requisition form." This fact is recorded as part of the process instance. Figure 7 illustrates the availability of this information by indicating a historic association between Task 1 and Dick. When the worklist for Task 4, "approve order," is generated, it will be recognized that a session with Tom acting as the accounts payable manager should not be allowed. It can be seen from the red lines in Figure 7 that if Task 4 appears on Tom's worklist, Tom and Dick would be able to conspire. Since Tom and Dick are identified as conflicting users, they are likely to conspire, and the risk should therefore not be taken.

Prototype implementation

The principles proposed in CoAP have been implemented in a prototype system. This system consists of two parts: an administration tool and a scaled-down workflow system. Both parts of the system were developed using Visual Basic**6 utilizing a Microsoft

Figure 7 Enforcing the business rule "an order may not be approved by its initiator" through conflicting users and conflicting tasks



Access database. This section first describes the results in terms of the end user. A brief discussion regarding implementation details follows. Finally, remarks about integrating the model in commercial off-the-shelf products are made.

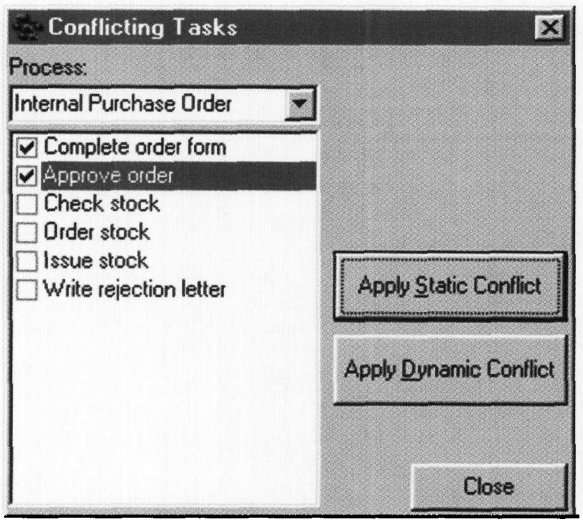
The end-user perspective. Figure 8 shows a screen that enables the administrator to identify conflicting tasks within a specific process definition. In Figure 8 the tasks "complete order form" and "approve order" were identified as dynamically conflicting tasks by applying the "apply dynamic conflict" option. For the purpose of our discussion we registered three users: Tom, Dick, and Harry. Since Tom and Dick are brothers, we identified them as dynamically conflicting users on a screen similar to Figure 8. All three of the users were identified as managers who normally would therefore be allowed to perform a wide variety of actions, including completing and approving internal purchase orders.

The internal purchase order process begins with a user completing a purchase order, after which it must be approved by a manager. Only then will stores issue or order the applicable items. This process was defined and an instance generated.

Tom completed a purchase order. Figure 9 depicts the worklists of the three users after this initial step. The following observations can be made:

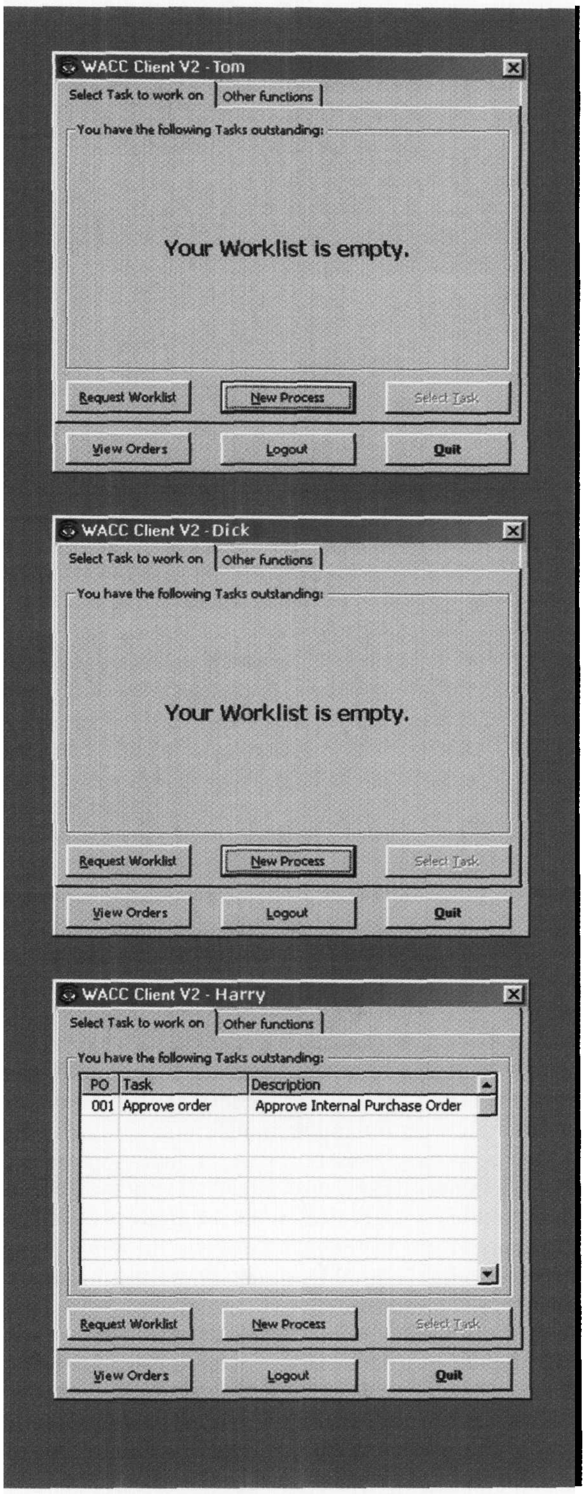
- Tom, being the creator of the order, may not approve the order. This observation is a direct result

Figure 8 Specifying conflicting tasks



- [✓] Complete order form
- [✓] Approve order
- [ ] Check stock
- [ ] Order stock
- [ ] Issue stock
- [ ] Write rejection letter

Figure 9 Tom, Dick, and Harry's worklists



PO	Task	Description
001	Approve order	Approve Internal Purchase Order

of the "complete order" and "approve order" tasks being indicated as dynamically conflicting tasks.

- Dick, Tom's brother, also may not approve the order. This observation is a result of Tom and Dick being identified as conflicting users, as well as the two tasks being conflicting.
- Harry, however, may approve the order since he did not create it, nor is he a conflicting user with Tom.

This example shows that it is possible to express a complex business rule through the consistent application of CoAP. CoAP provides an easy and intuitive way of expressing separation of duty requirements in the workflow environment.

Implementation lessons. The administrative component and the workflow component were implemented as separate projects. We briefly highlight implementation lessons and considerations from each.

The *administration tool* was developed to assist the security administrator with specifying high-level organizational separation of duty policy. The conflicts that are identified prohibit certain assignments according to Table 3. Consider the following example.

Assume that a new user-role association is made. The administration tool will first determine whether the user is involved in a conflicting user relationship and whether the role is involved in a conflicting relation-

Table 3 Static conflict interpretation

May Be Associated with	Users		Roles		Permissions		Tasks	
	Conflicting	Non-conflicting	Conflicting	Non-conflicting	Conflicting	Non-conflicting	Conflicting	Non-conflicting
Conflicting users			✗	✓				
Conflicting roles	✗	✓			✓	✓	✓	✗
Conflicting permissions			✓	✗				
Conflicting tasks			✓	✗				

ship. If both are nonconflicting, no further action is required, and the association can be made. However, if one or both are involved in a conflicting relationship, a check that is in line with Table 3 is performed to see whether the association should be allowed (✓) or disallowed (✗). Where appropriate, the administration tool will also suggest remedial action to disallowed associations. For example, if conflicting tasks are associated with nonconflicting roles, the tool will suggest that the roles are made conflicting. Similar checks are performed when two entities are made to conflict. Existing associations are evaluated to determine whether or not the new conflict will cause integrity problems. Again, the tool provides feedback regarding possible remedial action.

In the initial version of the administration tool, these checks were performed as part of the application code. However, in a bid to ensure integrity of the access control information regardless of the administration tool, we are expressing these constraints as ECA (event, condition, action) rules for active database technology.[29] Implementation of these rules as Oracle** triggers is currently underway.

The *run-time component* of the prototype, in the form of a scaled-down workflow engine, was developed to gain a better understanding of the functioning of workflow environments. At the same time, the proposed model was implemented. From the outset the intention was to develop the various components in a highly modularized fashion. To this end, four main modules exist at server level:

- A communication module, which establishes secure communication over a Transmission Control Protocol/Internet Protocol (TCP/IP) network via the use of DirectX** application programming interfaces.
- An access control module, which manages user sessions and grants permissions by allowing methods to be executed on objects.
- A module responsible for the workflow functionality that manages tasks and executes automated activities. The application module handles the work that a user performs and updates the application objects on the database.
- A database module, which is responsible for all storage requirements of the prototype. In this respect, the database is divided into three main parts: the access control database, the workflow database, and the application database. The access control database stores all user-related data, such as user lists, role assignments, and role hierarchies. The workflow database stores all the workflow-relevant data, e.g., routing information for documents and access control rules. Finally, the document database stores all the documents that are used by the business process.

The worklist is established by querying the database. Since the expression of the conflicts is explicit, the evaluation of the separation of duty constraints can be performed by means of a Structured Query Language (SQL) query without additional parsing of rules. This query forms the crux of the enforcement of access control requirements and, therefore, warrants further attention. This is best done by providing a pseudocode skeleton of the query (for establishing the worklist of user_x):

```
SELECT tasks FROM active_task_instances
WHERE user_x has accepted responsibility for the
    task
UNION
(SELECT tasks FROM active_task_instances
 WHERE user_x is of role senior to required role
MINUS
 SELECT tasks FROM active_task_instances
 WHERE user_x violates conflict)
```

In other words, a user will receive work items in his or her worklist that he or she already accepted, as well as work items based on tasks that require a role junior to the user's role. Tasks that will cause conflict if the user performs them are removed from the list. In the above pseudocode, the statement "user_x violates conflict" thus represents further subqueries that compare the user, the user's role, the tasks and the permissions of the tasks with the relevant conflicting sets and with the workflow history. Because of the amount of subqueries involved, the establishment of the worklist is a relatively expensive query. The alternative of maintaining a physical worklist for each user also presents huge overheads in large environments since the synchronization of the worklists with the active task instances would also be expensive. Further work regarding an optimal solution for the implementation of the worklist, specifically for deployment in large-scale systems, would thus be necessary.

The implementation of a stand-alone prototype afforded us the opportunity to investigate the implementation of separation of duty mechanisms in workflow environments. However, a pressing issue is whether the CoAP paradigm for the management of separation of duty requirements can be integrated in commercially available workflow management software.

Integrating CoAP in commercial environments. At this stage, no attempt has been made to implement CoAP in conjunction with a commercially available workflow system. However, some comments regarding such an implementation are in order.

Should the vision of the Workflow Management Coalition regarding a fully interoperable environment[2] be realized, the CoAP model will have an influence on two components: the administration tools and the worklist handler. However, currently the distinction between the different components is not clear. In this respect a commercial workflow management system will have to exhibit the following properties to enable the integration of CoAP:

- The administration side should be exposed to customization. Alternatively, the administration side should be separated from the rest of the system with a well-defined interface. Separation would allow the replacement of the current administration tools with tools based on CoAP.
- The creation of the worklist should be customizable. Should the worklist be maintained as a physical entity, database-level programming in the form of triggers might enable the appropriate removal of work items from the individual worklists.
- The timing of the granting and revocation of access rights possibly holds the biggest challenge in current systems. It would require the task abstraction to be clearly separated from the rest of the system. For example, in their SALSA prototype, Kang et al.[8] use a decentralized approach with no central workflow engine. Each task contains a small portion of the workflow specification since it knows with which tasks to interact. In such a distributed fashion, the task object itself controls the access to it, and the granting and revoking of access largely becomes irrelevant. The task objects need to interact with a monitor service to be able to enforce dynamic separation of duty requirements.
- In systems that do not support role-based access control principles, a role server may also be required to provide that functionality.[9] The role server will issue users with role certificates, based on user-role assignments and the role hierarchy, which then have to be evaluated by the workflow engine when the worklists are assembled.

From the brief comments above, it is clear that the integration of CoAP into a commercial workflow system is far from arbitrary. It presents interesting challenges that will have to be addressed through a variety of approaches.

Conclusion

This paper identified various kinds of conflicts that exist when considering separation of duty requirements in workflow environments. CoAP is suggested as a way to handle the administration of separation of duty requirements in workflow examples. With CoAP we identify users, roles, permissions, and tasks that together may result in fraudulent activities. The simplicity of the paradigm allows administrators to easily specify static and dynamic separation of duty requirements while maintaining complex relationships within the access control system.

It was further suggested that the additional requirement of "strict least privilege" suggests special attention be given to the design of role hierarchies and the association of tasks with those roles. Future work will concentrate on the role of engineering requirements for role-based workflow environments.

Movements in the area of workflow also introduce new and exciting challenges not addressed in this paper. In particular, the field of collaborative computing and workflow computing share several synergies and will, eventually, converge. Convergence brings about an interesting question regarding the task abstraction, which then may include collaborative tasks. There is also a demand for workflow systems to support work that is less structured and where all the enumerations of the paths cannot be exactly predicted.[30] This more *ad hoc* approach to workflow management will have a profound impact on the way in which we specify access control requirements.

**Trademark or registered trademark of Microsoft Corporation or Oracle Corporation.

Cited references and note

1. F. E. Allen, "Turning Points in Interaction with Computers," *IBM Systems Journal* **38**, Nos. 2&3, 135–138 (1999).
2. D. Hollingsworth, *The Workflow Reference Model*, WFMC-TC 1003, Issue 1.1, Workflow Management Coalition (January 1995); available from www.wfmc.org.
3. *Information Processing Systems—Open Systems Interconnection—Basic Reference Model—Part 2: Security Architecture*, ISO 7498-2, International Organization for Standardization (1989).
4. F. Leyman and D. Roller, *Production Workflow: Concepts and Techniques*, Prentice Hall, Upper Saddle River, NJ (2000).
5. The Workflow Management Coalition refers to activities but recognizes tasks as a synonym. See *Terminology & Glossary*, WFMC-TC1011, Issue 2.0, Workflow Management Coalition (June 1996); available from www.wfmc.org.
6. E. Bertino, E. Ferrari, and V. Atluri, "Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, 65–104 (February 1999).
7. J. Barkley, *Workflow Management Employing Role-Based Access Control*, U.S. Patent No. 6,088,679 (July 11, 2000).
8. M. H. Kang, J. S. Park, and J. N. Froscher, "Access Control Mechanisms for Inter-Organizational Workflow," *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies SACMAT 2001*, Chantilly, VA (May 3–4, 2001), pp. 66–74.
9. G.-J. Ahn, R. S. Sandhu, M. Kang, and J. Park, "Injecting RBAC to Secure a Web-Based Workflow System," *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, Berlin (July 26–28, 2000).
10. R. S. Sandhu, E. J. Coyne, H. L. Fenstein, and C. E. Youman, "Role-Based Access Control Models," *Computer* **29**, No. 2, 38–47 (February 1996).
11. J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE* **63**, No. 9, 1278–1308 (1975).
12. D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of IEEE Symposium on Security and Privacy* (April 1987), pp. 184–194.
13. R. Sandhu, "Transaction Control Expressions for Separation of Duties," *Proceedings of the 4th Aerospace Computer Security Conference* (December 1988), pp. 282–286.
14. R. Sandhu, "Separation of Duties in Computerized Information Systems," *Proceedings of IFIP WG11.3 Workshop on Database Security* (September 1990).
15. M. J. Nash and K. R. Poland, "Some Conundrums Concerning Separation of Duty," *Proceedings of the 1990 IEEE Symposium on Security and Privacy* (May 1990), pp. 201–207.
16. D. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations," *Proceedings of the 1995 Computer Security Applications Conference* (December 1995), pp. 241–248.
17. R. Simon and M. E. Zurko, "Separation of Duty in Role-Based Environments," *Proceedings of the 10th Computer Security Foundation Workshop*, Rockport, MA (June 10–12, 1997), pp. 183–194.
18. D. R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems," *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Fairfax, VA (October 1997), pp. 23–30.
19. V. D. Gligor, S. I. Gavrila, and D. Ferraiolo, "On the Formal Definition of Separation of Duty Policies and Their Composition," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA (May 3–6, 1998), pp. 172–183.
20. M. Nyanchama and S. Osborn, "The Role-Graph Model and Conflict of Interest," *ACM Transactions on Information and Systems Security* **2**, No. 1, 3–33 (February 1999).
21. G.-J. Ahn and R. S. Sandhu, "The RSL99 Language for Role-Based Separation of Duty Constraints," *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, Fairfax, VA (October 28–29, 1999), pp. 43–54.
22. R. K. Thomas and R. S. Sandhu, "Towards a Task-Based Paradigm for Flexible and Adaptable Access Control in Distributed Applications," *Proceedings of the 1992–1993 ACM SIGSAC New Security Paradigms Workshop*, Little Compton, RI (1993), pp. 138–142.
23. R. K. Thomas and R. S. Sandhu, "Conceptual Foundations for a Model of Task-Based Authorizations," *Proceedings of the IEEE Computer Security Foundations Workshop*, New Hampshire, IEEE Press (1994).
24. R. K. Thomas and R. S. Sandhu, "Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Authorization Management," *Database Security, XI: Status and Prospects*, T. Y. Lin and S. Qian, Editors, Chapman and Hall, London (1997), pp. 166–181.
25. S. Oh and S. Park, "Task-Role Based Access Control (T-RBAC): An Improved Access Control Model for Enterprise Environment," *Proceedings of the 11th International Conference on Database and Expert Systems Applications, DEXA 2000* (2000), pp. 264–273.
26. 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*, No. 26, 212–216 (November 2000).
27. V. Atluri and W-K. Huang, "An Authorization Model for Workflows," *Proceedings of the Fifth European Symposium on Research in Computer Security*, Rome, Italy, and *Lecture Notes*

in Computer Science, No. 1146, Springer-Verlag, Berlin (September 1996), pp. 44–64.
28. D. Cholewka, R. A. Botha, and J. H. P. Eloff, "A Context-Sensitive Access Control Model and Prototype Implementation," *Proceedings of the IFIP TC11 15th International Conference on Information Security (SEC2000)*, Beijing, China (2000), pp. 341–350.
29. N. W. Paton and O. Díaz, "Active Database Systems," *Computing Surveys* **31**, No. 1, 63–103 (March 1999).
30. C. Petrie and S. Sarin, "Beyond Documents: Sharing Work," *IEEE Internet Computing*, 34–36 (May–June 2000).

Accepted for publication April 30, 2001.

Reinhardt A. Botha *Faculty of Computer Studies, Port Elizabeth Technikon, University Way, Port Elizabeth 6000, South Africa (electronic mail: rbotha@computer.org).* Mr. Botha is a senior lecturer at the Port Elizabeth Technikon and a doctoral candidate at the Rand Afrikaans University. He has been involved in the design and implementation of electronic document management systems, as well as back-end systems integration since 1990. His main research interests include workflow, information security, and human-computer interaction. He received an M.Sc. degree from the Rand Afrikaans University in 1997.

Jan H. P. Eloff *Department of Computer Science, Rand Afrikaans University, P.O. Box 524, Auckland Park 2006, South Africa (electronic mail: eloff@rkw.rau.ac.za).* Dr. Eloff received a Ph.D. degree in computer science from the Rand Afrikaans University. He gained practical experience by working as a management consultant specializing in the field of information security. Since 1988 he has been a full professor in the Department of Computer Science at the Rand Afrikaans University. He is chairperson of the Special Interest Group in Information Security, affiliated with the Computer Society of South Africa. He is also chairperson of the International Working Group 11.2 of IFIP specializing in small systems security. He has published extensively in a wide spectrum of accredited international subject journals. Many acclaimed international and national conferences were organized and conducted under his leadership, and he has presented papers at leading information security conferences on an international level. Dr. Eloff is an evaluated researcher from The National Research Foundation (NRF), South Africa, and he is an advisor to industry on various information security projects.