A Context-sensitive Access Control Model and Prototype Implementation

Damian G. Cholewka[1], Reinhardt A. Botha[2], Jan H.P. Eloff[1]
[1] *Rand Afrikaans University, Johannesburg, South Africa*
[2] *Port Elizabeth Technikon, Port Elizabeth, South Africa*

Key words: Access Control, Role-based Access Control, Context-sensitive, Workflow

Abstract: Role-based access control associates roles with privileges and users with roles. Changes to these associations are infrequent and explicit. This may not reflect business requirements. Access to an object should not only be based on the identity of the object and the user, but also on the actual task that must be performed, i.e. the context of the work to be done. Context-sensitive access control considers the actual task when deciding whether an access should be granted or not. Workflow technology provides an appropriate environment for establishing the context of work. This paper discusses the implementation of a context-sensitive access control mechanism within a workflow environment. Although the prototype represents scaled-down workflow functionality, it illustrates the concept of context-sensitive access control.

1. INTRODUCTION

Current access control mechanisms frequently burden the end-users with unnecessary security related tasks. It may, for example, be expected from a user to assume a specific role at the beginning of a session, resulting in unnecessary multi-logons. Alternatively a user may automatically play the most senior role that he may hold and consequently receive the permissions associated with that role. The user is therefore trusted to implement the security policy and not misuse granted privileges.

The principle of context-sensitive access control is introduced to address these issues. Within the prototype the approach is taken that the dynamic

granting of access rights is designed into the workflow. The workflow infrastructure is thus directly responsible for the granting of access rights.

2. THE WORKFLOW ENVIRONMENT

A "handle an insurance claim" workflow is used as a case scenario throughout the paper. We consider the typical stages of a workflow, namely: definition and enactment [2].

The "handle an insurance claim" workflow's ***definition*** is graphically depicted in Figure 1. It is called a *process definition,* essentially being a collection of tasks. A task is the smallest unit of work, which must be done to achieve a certain business objective.

A task is defined by a *task definition* that specifies which work needs to be done on which objects, by which users. The "handle an insurance claim" process is defined as consisting out of 10 tasks, labeled ① – ⑩ in Figure 1. Task ① involves the initialization of the whole process by generating a new document called a claim schedule. Initially a claim schedule may consists of a claim form, a police report, witness reports and quotations. This task is to be completed by a clerk. Task ② follows on completion of task ①. Task ② is an automated decision made by the system based on workflow control data, in this case the claim value that was captured as part of task ①.

The process definition also contains *business rules* that specify conditions for executing tasks. In the example, route ②-⑦ will be followed if the claim's value does not exceed 5000. Route ②-③ will be followed if the value of the claim exceeds 5000. The other tasks and rules depicted in Figure 1 follow the same convention.

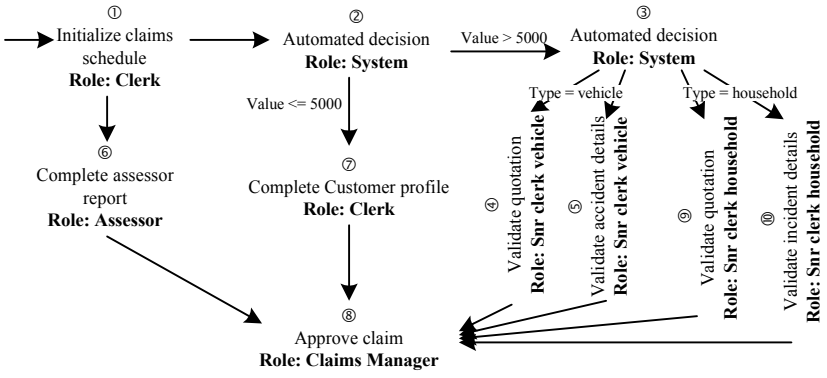

Figure 1. The "handle insurance claim" process definition

The ***enactment*** phase of the workflow is concerned with the performance of the actual business task.

A user interacts with the workflow system through a *worklist* that supplies the user with the *work items* that he has to complete. A work item represents a task instance assigned to the user by the workflow engine. In the example, the worklist of a manager would contain a list of all the claims to be approved.

Many *process instances* can thus be instantiated from the same process definition. The "handle an insurance claim" process definition will be used to create a process instance for every claim that is submitted. Each process instance will generate *task instances* based on the task definitions in the corresponding process definition. Process and task instances will use associated *workflow control data,* such as the value of the claim, to evaluate the rules in the process definition. Now consider the access control requirements of the described workflow.

3. ACCESS CONTROL REQUIREMENTS

The access control requirements of the workflow environment are characterized by three concepts. Firstly, the access rights granted should be determined by the task to be performed. The workflow environment imposes a specific *sequence of events* in that certain conditions must be met before the next task can begin. Secondly, *strict least privilege* dictates that the access rights should be the absolute minimum required for the task. Finally, the access control mechanism should support *separation of duty* policies that may be required. Each of these requirements is now discussed in more detail.

3.1 Sequence of events

The granting of certain privileges depends on the successful completion of other tasks. For example, a claim may not be approved unless a customer profile has been completed. The mechanism should therefore be able to recognize at which point it is within the execution of a claim. Current access control mechanisms do not recognize any difference between an "approve claim" now and an "approve claim" later. [6].

3.2 Strict least privilege

The concept of least privilege usually implies that a user is assigned the minimum privileges necessary to perform his *job* [4]. A manager will

therefore receive privileges to perform tasks that he must do at some stage of his job. The fact that some of these privileges may be superfluous for many of the daily *tasks* is recognized by introducing the concept of strict least privilege. It strengthens the least privilege concept to restrict privileges to the minimum rights required *at a specific point in time*, *for a specific task.*

In the "handle an insurance claim" example this would imply that a manager who "initialize claim schedule" should only receive the "clerks" rights which are necessary to perform the task, whilst rights like *approve* or *deny* claim are not granted.

3.3 Separation of duty

Separation of duty (SoD) is a security principle used to formulate multi-person control policies [5]. In essence it requires that two or more different people are responsible for the completion of a business process. It would thus, in principle, discourage fraud by requiring a conspiracy, thereby increasing the risk to the potential perpetrators.

Static SoD [3] requires that the membership to two roles must be strongly exclusive. In the "handle an insurance claim" workflow Static SoD requirements can be enforced by using disjoint role hierarchies. For example, the assessor and manager are in different role hierarchies; furthermore, an assessor's and a manager's privileges are disjoint. The process definition thus requires both a manager and an assessor to be involved.

Dynamic SoD policies provide increased flexibility by controlling the activation and use of roles [3]. The strict least privilege requirement essentially fulfills a dynamic SoD requirement since it does not restrict the potential of a user to belong to more than one role, but forces a user to only have one role active at a specific point in time [5]. [1] and [5] enumerate several variations of dynamic SoD.

Dynamic SoD policies can only be enforced through evaluation at execution time [3]. For example, in the "handle an insurance claim" workflow a dynamic SoD requirement which stipulates that a manager may not approve a claim that he himself initialized can only be evaluated at execution time. This requirement cannot be evaluated without considering the "handle an insurance claim" process definition. It must be specified that task ① and task ⑦ may not be performed by the same user.

4. MODEL DESIGN

The context sensitive access control model is based on the RBAC96 reference model. It has been pointed out by Sandhu et al [4] that the RBAC96 model supports principles such as least privilege and separation of duties, but that it does not enforce the *use* of such principles. The context-sensitive access control model proposed here does not attempt to replace RBAC. To the contrary, it builds on the RBAC foundation in order to provide in additional needs. The main components of the proposed model are depicted in Figure 2.

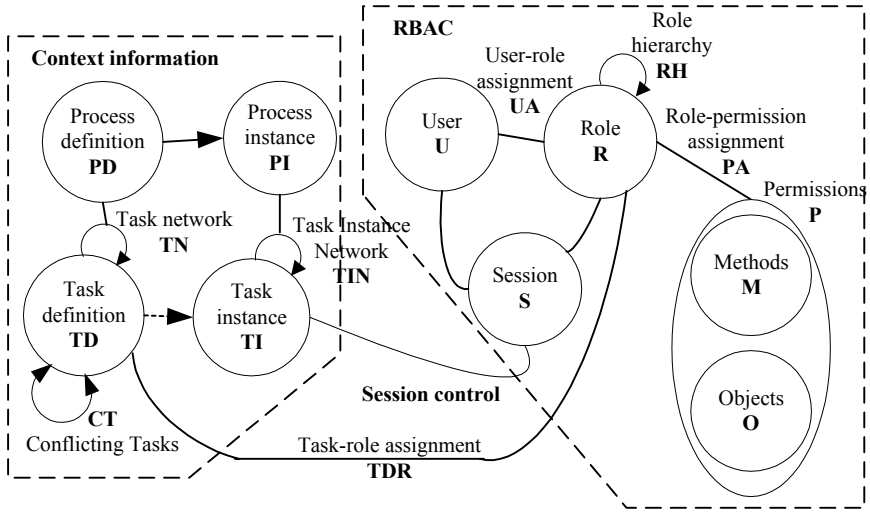

Figure 2. *The context-sensitive access control model*

4.1 RBAC

In line with RBAC96 definitions, users (U) belong to roles (R) as defined by a user-role assignment relation (UA). The permissions (P) that are associated with a role are reflected in the role-permission assignment relation (PA).

The permission abstraction (P) of the RBAC96 model has been interpreted in this environment as reflecting the available methods (M) for an object (O). A user therefore could receive, for example, the permission to execute the ApproveClaim method for a claim form object.

A user receives the permissions associated with role(s) that he assumes for the session (S). The RBAC96 model is not prescriptive with regards to the interpretation of a session. It merely sees a session as a time-bound construct to associate users, roles and permissions. The essence of the proposed model lies in the interpretation of the session construct.

4.2 Context information

The context of the work is provided through reference to the process definition (PD). A process definition consists of multiple task definitions (TD) arranged according to a task network (TN).

The conditions specifying the routes to be followed are maintained as part of the task network. The role(s) capable of performing a specific task is identified through the task-role assignment relation TDR.

The process instance (PI) relation keeps track of all instantiated processes. Since some tasks are not instantiated due to the conditional nature of the workflow routes, the task instance (TI) relation is used to keep track of the task instances. Task instance network (TIN) relates the task instances in a similar fashion to how TN combines various task definitions.

Separation of duty is implemented in the current model as conflicting task sets (CT). If two tasks belong to a conflicting task set, the two tasks must be performed by different users. From an access control perspective the most important design decision was to redefine the concept of a session.

4.3 A session control approach

Traditionally sessions indicate the elapsed time between when a user logs on to the server until he logs out. Privileges in these traditional sessions are relatively static over time, although a user may have different commands at his disposal to change roles.

The basic approach of the proposed model in this paper is to use any one session for one and only one task. Although a user may authenticate once to the system, he initially receives no privileges. When a user acts on a work item in the worklist, a workflow session is established and the appropriate rights become available to the user. As soon as the user stops working on that specific task the workflow session is closed and all the associated privileges are revoked. This concept of a session is graphically depicted in Figure 3.

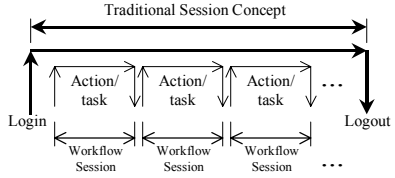

Figure 3. The revised session concept

A session is thus used to control the permission of a user for the duration of activity on a task. A session maps to one and only one role.

5. PROTOTYPE DEVELOPMENT

A small, scaled-down workflow kernel was developed which is responsible for the propagation of work through the interpretation of basic routing instructions.

A separate module in the prototype is responsible for the interpretation of access permissions as specified in the workflow definition. Context-sensitive access control is enforced by addressing the three aforementioned access control requirements.

- S*equence of events*. A user receives different access rights to the same object depending on the exact task that he is performing.
- S*trict least privilege*. Privileges are not only different over time, but are also the absolute minimum required to complete the task.
- *Separation of Duty*. The principle of conflicting tasks introduced dynamic SoD constraints to the workflow prototype.

The prototype consists of a client component and a server component. Both components are developed in Visual Basic 6. The client component provides the interface with the user in the form of a worklist and a forms environment, whilst the server component represents the workflow engine and data storage. Communication is based on DirectX7. The database was developed in MS Access97. All access to the database is regulated exclusively through pre-defined SQL queries stored in the database.

6. AN EXAMPLE WORKFLOW ENACTMENT

The prototype's operation is described by considering the main activities that can occur within the system. The workflow example presented in Figure 1 is used. In particular, the enactment will instantiate "Claim001", a household claim to the value of 3500.

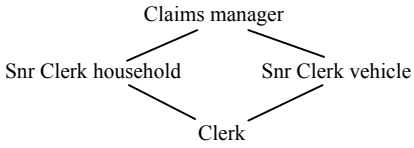

Figure 4. Role hierarchy

Abel, Grant and Frans are used as users throughout the discussion. Abel and Grant are claims managers, whilst Frans is an assessor. The claims manager role forms part of a role hierarchy depicted in Figure 4. The assessor role does not form part of this role hierarchy.

6.1 Initiating a new claim

When the user initiates a new "handle claim" process, some important events take place. Firstly a new process instance (PI) is created from the process definition template (PD). The first task that needs to be performed as part of the process is identified through evaluation of the task network (TN). Thereafter the new task instance (TI) is created from the task definition template (TD). The task instance is marked as inactive.

The worklist of a specific user is populated through a query that combines the active tasks already accepted by that user with the tasks that are still inactive. It determines whether the required role for the inactive tasks can be assumed by the user and whether all SoD requirements are met. If all the conditions are true, then the task item will be listed in the worklist.

It is important to realize that up to this stage the user has not received any permission. The user still has no active role – all decisions up to now have been based on potential roles, not active roles.

When a user selects to work on an item the access control module determines the role needed for that specific task. Consider the worklist of Abel, being a claims manager. She can assume the role claims manager which is senior to clerk. However, if Abel chooses to perform the "Initialize claim schedule" work item for "Claim 001", she will assume the role clerk and therefore only receive the privileges associated with the clerk role. At this stage there are no SoD constraints to consider.

Once Abel selects to "Initialize claim schedule" for "Claim001", the task's state changes from "Inactive" to "Busy". She is then presented with a form on which to enter the claim details. This corresponds with task ① in Figure 1. On completion of the task the state of that specific task changes to "Completed".

At this point all requirements are met for task ② to be executed. Since task ② is an automated decision, the workflow engine executes the conditional split and creates tasks ⑥ and ⑦ in TI.

6.2 Complete Customer Profile and Assessor Report

In the example a SoD requirement is specified by identifying tasks ① and ⑦, as well as tasks ① and ⑧ as conflicting tasks. The worklists of Abel, Grant and Frans are shown in Figure 5. Note that Abel does not receive the option to "Complete Customer Profile" because of the dynamic SoD constraint between tasks ① and ⑦. Frans, being an assessor may only perform the "Complete Assessor report" task. This shows how static SoD can be achieved through disjoint role hierarchies. Grant may perform

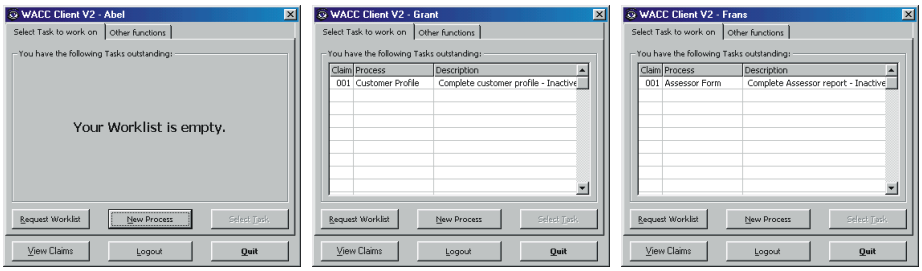

Figure 5. Worklists of Abel, Grant and Frans after Abel initialized claim

"Complete customer profile" (task ⑦), since he did not initialize the claim (task ①).

Once Grant selects to work on the "Complete customer profile" task, he is presented with the Complete Customer Profile section of the claim form as illustrated in Figure 6. Note that he does not receive any unnecessary privileges regarding the claim form. The state of the task changes to "Busy". Should he not complete the task, the state changes to "Wait". Once he submits the form, the task will change status to "Completed". Permissions are only granted while the task is in the "Busy" state.

Task ⑥ in our example has to be completed by an assessor. Once Frans completes task ⑥, the workflow kernel can do a merge operation and create an instance of task ⑧, namely "Approve claim". This will then appear in the worklists of users who may approve the claim. In this case it will only be claims managers. However, although Abel is a claims manager she will not be able to approve the claim (task ⑧) since she initialized it (task ①).

6.3 Approve claim

When Grant chooses to approve the claim he is presented with the screen in Figure 7. Note that Grant now has considerable different access rights to what he had in Figure 6. When Grant approves the claim, the state of the task changes to "Completed".

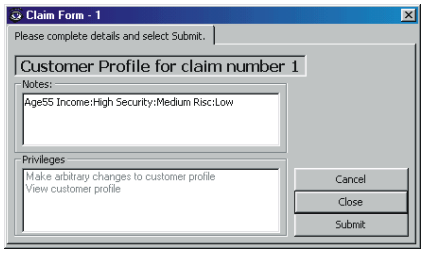

Figure 6. Grant completes customer profile

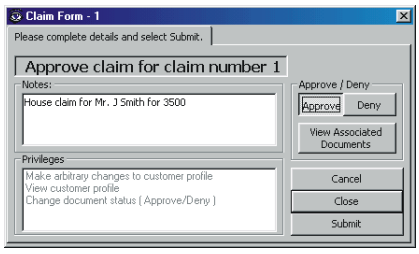

Figure 7. Grant approves claim

Since Task ® is the last task in this "Handle an insurance claim" process definition it concludes the process and the claim is completed.

7. CONCLUSION

The prototype serves to illustrate and explain the concept of context-sensitive access control. It demonstrates the influence of sequence of events on the access control decision. Users receive the minimum access and one dynamic SoD constraint requirement was implemented. Further work regarding SoD constraints in the workflow environment is necessary.

Due to the development environment chosen for the prototype, the approach hinges strongly on relation database principles. We believe that certain advantages could be achieved by a more object-oriented approach. The vision is that each object (e.g. claim) will make its own decisions with respect to who may receive what kind of access to it based on the current circumstances. This would result in a solution that is more suitable for heterogeneous systems, as well as adaptive workflow environments.

The prototype can benefit from the use of a database with active components such as triggers. Many of the functionality now residing in the server module could then be integral in the data repository. This would be closer to full object autonomy than what is currently the case.

REFERENCES

[1] V.D. Gligor, S.I. Gavrila and D.Ferraiolo. On the Formal Definition of Separation of Duty Policies and their composition. Proc IEEE Symposium on Security and Privacy, May 1998.

[2] D. Hollingsworth. The Workflow Reference Model. Document Number TC-00-1003. Issue 1.1. 29 Nov 1994. www.wfmc.org

[3] D.R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. Proc 2nd ACM Workshop on Role-based Access Control, Fairfax, VA, Oct 1997.

[4] R.S. Sandhu, E.J. Coyne, H.L.Fenstein and C.E. Youman. Role-based Access Control Models. IEEE Computer, 29(2), Feb 1996, 38 – 47.

[5] R. Simon and M.E. Zurko. Separation of duty in Role-based Environments. Proc of 10th Computer Security Foundation Workshop, Rockport, Massachusetts, 10–12 Jun 1997.

[6] R.K.Thomas and R.S. Sandhu. Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. Proc IFIP WG11.3 Workshop on Database Security, Lake Tahoe, California, 11 – 13 August 1997.