ACCESS CONTROL AND SEPARATION OF DUTY IN AGENT-BASED WORKFLOW ENVIRONMENTS

Willem Engelbrecht, Reinhardt Botha and Rudi Harmse
Faculty of Computer Studies,
Port Elizabeth Technikon, Port Elizabeth
willemengelbrecht@webmail.co.za, rbotha@computer.org, rudi@petech.ac.za

Abstract

Agent Technology provides a new methodology in implementing workflow environments. This paper is concerned with how this shift in paradigm affects traditional security concepts like access control and separation of duty principles. The discussion focuses on the implementation of task allocation in an agent-based workflow environment (AWE) that is currently being developed. Task allocation is fundamentally driven through constraining access of users through enforcing constraints.

Keywords: Agents, Access Control, Separation of Duty, Task Allocation, Dynamic Workflow Environments

1. INTRODUCTION

Agent technology has revolutionized the shape of distributed environments by fundamentally combining the functionality of the servlet and applet. The result of this merger is the mobile agent paradigm.

Mobile agents are software abstractions that can migrate across the network representing users in various tasks [6].

As workflow environments primarily exist in distributed environments, the natural integration between these two technologies is obvious. The result of this integration is an agent-based paradigm, which allows for greater flexibility and robustness.

Autonomous agents and multi-agent systems represent a new way of analyzing, designing and implementing complex software systems. The agent-based view offers a powerful repertoire of tools, techniques and

metaphors that have the potential to considerably improve the way in which people conceptualize and implement many types of software [3].

The context for this paper is an *agent-based workflow environment* **(AWE)** that is being developed for the purpose of creating a dynamic workflow environment. In this paper the discussion will focus on how two traditional security principles, namely access control and separation of duty, are affected by this shift in paradigm.

2. AWE: AN AGENT-BASED WORKFLOW ENVIRONMENT

The proposed agent-based workflow environment (AWE) is based on the concept that business objects control the execution of the environment.

It is this realization that led to the creation of the Agent Business Object **(ABO)** which is the union of agent technology, business objects and business logic. This ABO is the vehicle that allows the environment to be dynamic.

As depicted in Figure 1, an ABO primarily moves between two states **(instantiated, deployed)** in the AWE environment. As a business document enters the queue it gets identified. Based on the identification of the document an ABO is instantiated and the supporting components for that process will be added to the ABO. The ABO is then deployed and initiates its own internal workflow engine.

Figure 1 depicts an abstract view of the AWE environment, from which the following five main components of an ABO can be identified:

- The **Core** houses all common capabilities, that are shared between agent business object instances. This includes the ability of the agent to traverse the network, invoke services from other agents and its own internal workflow engine.
- The **Tasks** component houses the specifications used in an ABO workflow engine for a specific process. This includes the sequence in which tasks must be preformed.
- Any **Rules** that the workflow engine needs to adhere to for a specific process will be defined in this component. This constitutes the constraint-base.
- The **Data** component stores all the information that is generated or required throughout the execution of the ABO.
- The **Users** component stores a resolved Task-User Matrix for the specific process.

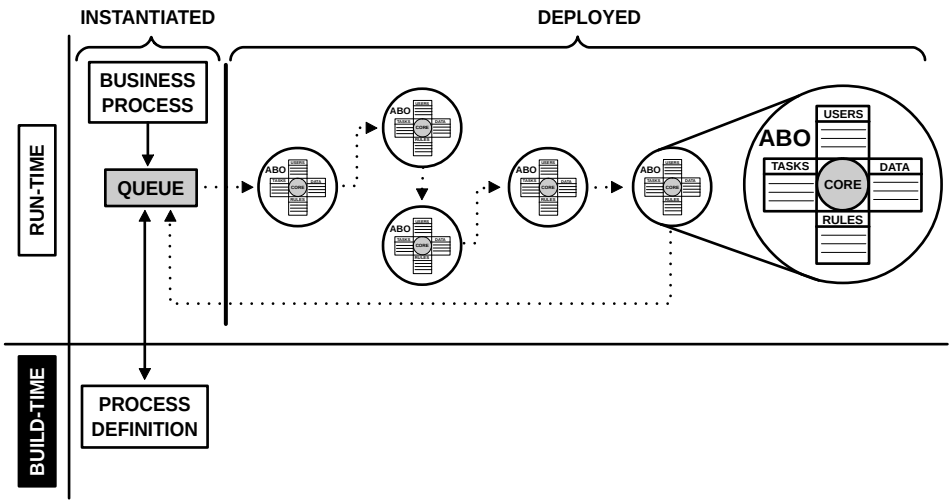

Figure 1 An abstract view of the AWE environment

It is the interaction between these components that allows the security principles to be implemented effectively.

3. ACCESS CONTROL CONSIDERATIONS

Access Control is fundamentally concerned with the question: "Is this individual allowed access to a specific object?" Currently numerous variations on implementing access control exist, but combined their goal is to answer the above question more efficiently.

Role-based access control is being widely accepted not only as an access control policy but also as a flexible permission management framework in various commercial environments [5].

3.1. Role-Based Access Control

The central notion of role-based access control (**RBAC**) is that permissions are associated with roles, and users are assigned to appropriate roles. Roles are created for the various job functions in an organization and users are assigned to roles based on their responsibilities and qualifications [7].

With RBAC, security is managed at a level that corresponds closely to the organization's structure [4].

RBAC is a mechanism that can implement a variety of policies. Separation of duty policies are often closely tied to RBAC models, because separation of duty is critical in many commercial applications [4].

3.2. Separation of Duty

The purpose of separation of duty is to prevent fraud by requiring the involvement of more than one individual in completing a process.

When implementing separation of duty using constraints, two types of mutual exclusion are considered, authorization-time exclusion and run-time exclusion. Authorization-time exclusions are mutual exclusion rules that are applied at role-authorization time. Run-time exclusions are applied at run-time during a user session. These forms of exclusions have been termed static and dynamic separation of duty [4].

- **Static** separation of duty requirements can be satisfied by the assignment of individuals to roles. Static separation of duty requires that the same user cannot be made a member of mutually exclusive roles [5].
- **Dynamic** separation of duty allows one user to be a member of mutually exclusive roles, but places constraints on the simultaneous activation of roles [5].

To enforce the separation of duty requirements, although an integrity requirement, relies on an access control service that is sensitive to the separation of duty requirements [2].

Essentially separation of duty requirements are implemented by constraining users access.

3.3. Constraining Access

Constraints on role and user assignments to tasks in a workflow are at the heart of implementing RBAC and separation of duty.

Three main categories exist according to the time at which constraints can be evaluated [1], namely:

- **Static** constraints can be evaluated without executing the workflow.
  Task T2 has to be preformed by role R3.
- **Dynamic** constraints can be evaluated only during the execution of the workflow, because they express restrictions based on the execution history of the workflow.
  If four activations of task T1, within the same execution of the workflow, is

aborted by the same user, then that user is not allowed to execute task T1 anymore.

- **Hybrid** constraints are constraints whose satisfiability can be partially verified without executing the workflow.
  If a user belongs to role R1 and has performed task T1, then he/she cannot perform task T4. This constraint is also an implementation of dynamic separation of duty principles.

The aforementioned access control considerations will be demonstrated through the scenario of a tax refund process.

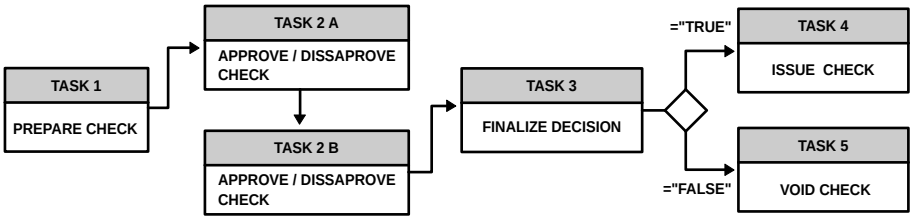

Figure 2 Workflow: Tax Refund

4. SCENARIO: TAX REFUND PROCESS

The **Tax Refund** process is defined as consisting out of 5 tasks as depicted in Figure 2. Task–1: Prepare Check requires the user to gather all information regarding the application for a tax refund. This task is to be completed only by a clerk. If a user cancels this task, for more than four times within the same execution of the workflow, then that user is not allowed to perform this task again.

Task–2: Approve/Disapprove Check follows on the completion of Task–1. Task–2 is duplicated into Task–2A and Task–2B for the purpose of serializing the workflow. Serialization was chosen for its simplicity in implementation, in principal the result of the aforementioned approach would be identical to a parallel execution.

Task–2A and Task–2B requires the users to review the information gathered in Task-1. These tasks must be preformed by two separate users belonging either to the role of manager or director. If either of the users disapproved the check it must earmarked to be voided.

Task–3: Finalize Decision follows on the completion of Task-2B. This task requires the user to review the decision made by the users that has preformed Task–2A and Task–2B to ensure that it is valid, thus finalizing the decision. This task is to be completed only by a user

that belongs to the role of director or higher, who have not taken part in Task 2A or Task 2B.

Based on the decisions of the users that completed Task–2 the check will either be voided or issued.

Task–4: Issue Check requires the user to issue the check. The user that performs this task cannot have any relation to the user that completed Task–1. This task is to be completed by a clerk or a role higher on the role order.

Task–5: Void Check is an automated task and the capabilities to perform this task is stored in the ABO. This task only requires that the decisions that have been made must be stored.

Constraints specific to this process are defined in Figure 3. It must be noted that in a real world example there could be additional constraints.

The specific semantics for defining the constraints is not important for this paper, therefore we use an intuitive constraint language as follows:

- `T1 DO BY ONLY[CLERK]`– Task–1 must only be done by a user of the role clerk.
- `U !DO T1 & T4` – A user must not do Task–1 and Task–4.
- `{U DO T4} !REL {U DO T3}` – A user that preforms Task–4 must have no relation to the user that preformed Task–3.

5. TASK ALLOCATION

Conceptually Figure 1 depicted how AWE will implement its access control policy. Initially the entire role hierarchy, task definitions, workflow specifications of each process and user assignments for the environment will be defined during build-time. As a business object enters the AWE environment at run-time it will be identified and the process of preparing the ABO for deployment will commence. The queue is responsible for generating the appropriate supporting components for the process that will be used in the ABO. This starts the process of static task allocation.

5.1. Static Task Allocation

The Static Task Allocation process is divided into two phases.

Phase 1 is concerned with applying the **static** constraints as depicted in Figure 3. To this end, the Role-user hierarchy as depicted in Figure 4 is used as it was defined at run-time. To demonstrate how static constraints are applied, consider the static constraint `T1 DO BY ONLY[CLERK]`.

The constraint specifies that Task–1 can only be completed by a user of the role clerk. From Figure 4 if can be observed that there are three

CONSTRAINTS : TAX REFUND		
STATIC	**HYBRID**	**DYNAMIC**
T1 DO BY ONLY [CLERK]	U !DO T2 A & T3	$\{U \text{ ABORT } T1\}_{4X}$ U !DO
T2 DO BETWEEN [MANAGER] & [DIRECTOR]	U !DO T2 B & T3	
T3 DO BY [DIRECTOR]	U !DO T1 & T4	
T4 DO BY [CLERK]	U !DO T2 A & T2 B	
	{U DO T4} !REL {U DO T3}	

Figure 3 Tax Refund: Constraint Base

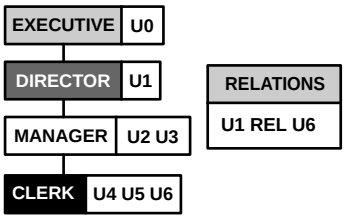

Figure 4 Tax Refund: Role Hierarchy

TASK 1	USER 6	USER 5	USER 4	
TASK 2 A	USER 3	USER 2	USER 1	
TASK 2 B	USER 3	USER 2	USER 1	
TASK 3	USER 1	USER 0		
TASK 4	USER 6	USER 5	USER 4	USER 3
	USER 2	USER 1	USER 0	

Figure 5 Task-User Matrix

users assigned to the role clerk namely, User–4, User–5 and User–6. These users are then assigned to Task–1. The result of applying all the static constraints that have been specified in the constraint base is the Task-User Matrix as depicted in Figure 5.

The Task-User Matrix and the constraint base depicted in Figure 3 are used as input for Phase 2.

Phase 2 is concerned with enforcing the **hybrid** constraints of the constraint base. Figure 6 depicts the process whereby the hybrid constraints are enforced on the Task-User Matrix.

The task with the lowest user count is selected as the starting point.

In the example in Figure 5 this is **Task–3** with only two users. As there are more than one user, the selection will be based on the workload of the users. We will assume User–1 has the lightest workload. From the constraint base depicted in Figure 3 it can be determined that there are two constraints that applies to task–3.

The first states that if User–1 performs Task–3, he/she cannot perform Task–2A or Task–2B, thus User-1 is removed from these conflicting tasks. As there are remaining users at these tasks, phase 2 can continue.

The second constraint that applies is that the user that performs Task–3 can have no relation to the user that performs Task–4.

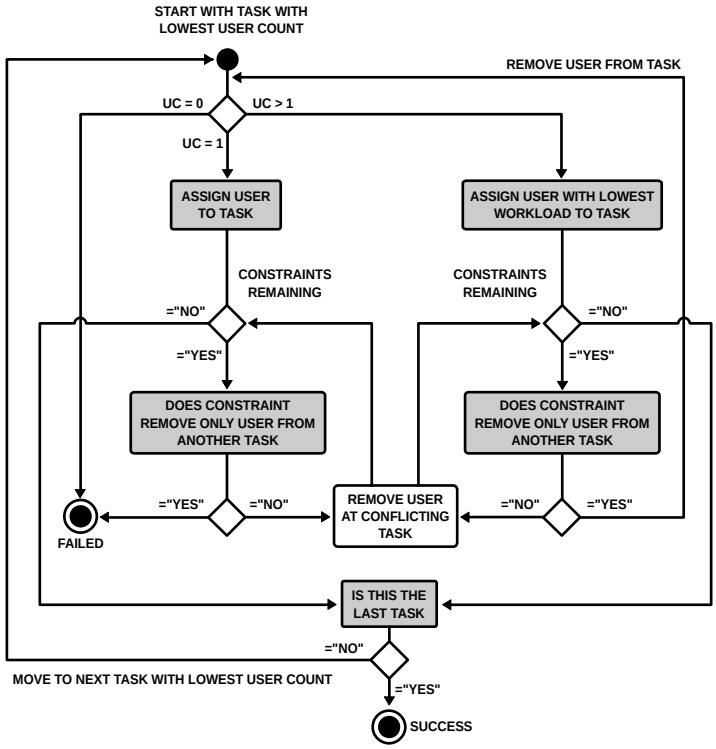

Figure 6 Process Algorithm for Task Allocation Phase: 2

From Figure 4 it is observed that User–1 and User–6 are related. As can be seen in Figure 5, User–6 is authorized to perform Task–4. The constraint can therefore be enforced by removing User–6 from Task–4. Since there are remaining users at Task–4, phase 2 can continue.

As all constraints that are specified for Task–3 have been successfully applied, phase 2 now moves to the next task, being the task with the lowest user count.

In the scenario this task is **Task–2A** which only has two users. We will assume that User–3 has the lightest workload. From the constraint base two constraints can be identified, namely that a user cannot perform both Task–2A and Task–2B. User–3 is thus removed from Task–2B because there are still users remaining at Task–2B the constraint is valid.

The next constraint is that a user cannot perform Task–2A and Task–3. This constraint was enforced when Task-3 was resolved. The user that performed Task–3, namely User–1 was removed from the Task-User Matrix at Task–2A. Thus selecting the user with the lightest workload namely User–3 would be sufficient to enforce this constraint.

The task which now has the lowest user count is **Task–2B**, with only one user assigned, namely User–2. The process depicted in Figure 6 indicates that when the user count is 1 (UC = 1), the user must be assigned, regardless of their workload. Nonetheless the constraints still apply.

From the constraint base two constraints can be identified. The first constraint states that a user cannot perform both Task–2B and Task–2A. Since Task–2A is resolved, and the user that performed Task–2A was removed as a valid choice in the Task-User Matrix for Task-2B the constraint is enforced.

The second constraint specifies that a user cannot perform both Task–2B and Task–3. Since Task–3 is resolved, the constraint is enforced, because the user that performed Task–3 was removed as a valid choice in the Task-User Matrix for Task–2B. Thus User–2 is a valid choice.

The next task to be resolved is **Task–1** with three users. We will assume that User–4 has the lightest workload. From the constraint base it can be identified that there is only one constraint that applies, namely that a user cannot perform both Task–1 and Task–4. Thus User–4 is removed from Task–4. As there are remaining users at Task–4 the constraint is validated. Thus User–4 is assigned to Task–1.

Task–4 is the final task to be resolved. From the constraint base it can be identified that there is only one constraint that states a user cannot perform Task–1 and Task–4. As Task–1 has already been assigned, this constraint was enforced and the user that performed Task–1 namely User–4 was removed as a valid choice to perform Task–4. We will assume User–5 has the lightest workload. Thus User–5 is assigned to Task–4.

Because all tasks has been assigned a user, Phase 2 is successful.

This **resolved** Task-User Matrix is then injected into the ABO. The performance of the agent business object's own internal workflow engine is improved by not being burdened with the process of resolving the role-hierarchy for each task after its deployment and by validating the Task-User Matrix, improving the stability of the execution.

5.2. Dynamic Task Allocation

Dynamic Task Allocation is enabled through the activation of dynamic constraints. From the constraint base it can be identified that there is one dynamic constraint. The constraint specifies that if four activations of task T1, within the same execution of the workflow is aborted by the same user, then that user is not allowed to execute task T1 anymore.

If this constraint was activated through the actions of User–4 the ABO will return the Task-User Matrix to its original state, where it will

enforce this constraint by removing User–4 as a valid user from Task–1. The ABO will not redo any tasks that have been successfully completed to this point. Unfortunately the constraint applies to Task–1, thus no other task has been completed. It will then internally resolve the Task-User Matrix as defined in phase 2 of static task allocation.

In the event that this dynamic constraint results in the tax refund process failing, the ABO will return to the Queue where it will raise an exception.

5.3. Optimizing Task Allocation

Each time a process, for example the tax refund process, is successfully completed the resolved Task-User Matrix for that process is stored. Thus creating a set of users that were involved in the execution of each task in the process.

Figure 7 depicts three of these sets (A,B,C). Thus at the completion of every instance of the process, the set will be merged with the collective. It must be remembered that this collective is only valid as long as the constraints that apply to the process have not changed. If a new user is added, it will only affect the collective by spawning a new path for this user. As this user is also subjected to the same constraints that were applied to the existing users. Thus allowing the constraint base to be logically enforced. A path through the collective is now only dependant on the users' individual workloads.

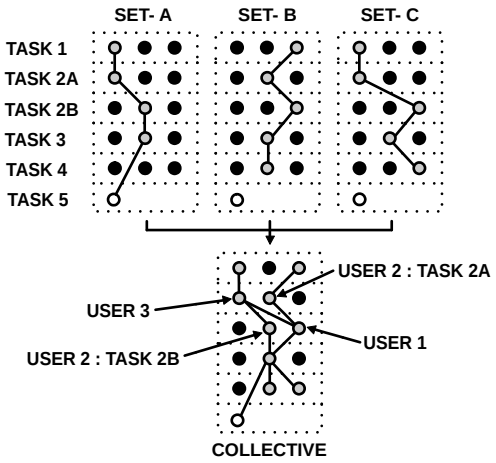

Figure 7 Optimization of Task Allocation

The collective can only be used as a alternative to phase 2 of static task allocation once it has reached a point of sufficient saturation.

This approach also allows exceptions to be handled efficiently. In the event that User–2 at Task–2B cannot complete the task, one only needs to move to the parent node of User–2, namely User–3, and it can immediately identified that User–1 is a valid alternative to User–2 as depicted in Figure 7. Thus the process will then continue at User–1.

By providing sample workloads for each user in the Task-User Matrix for a specific process, a collective can be pre-generated without having to deploy the ABO. Thus reducing the time it takes to sufficiently saturate the collective.

6. CONCLUSION

AWE implements the concept of separation of duty, through the ability of the agent business object, to dynamically assign tasks to the users at run-time. Thus constraints can be placed to ensure that an individual is not allowed to initiate and approve a process.

The fact that the agent business object is also responsible for selecting the next user, based on his/her respective workloads, exponentially increases the number of individuals required to perform a successful conspiracy.

References

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

[2] Botha, R. and Eloff, J. (2001). Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666 – 682.

[3] Jennings, N., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1:275 – 306.

[4] Kuhn, D. (1997). 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*, pages 23 – 30.

[5] Lee, H. and Noh, B. (1999). An integrity enforcement application design and operation framework in role-based access control systems: A session-oriented approach. *Proceedings of the 1999 International Workshops on Parallel Processing*, pages 179 – 184.

[6] Milojicic, D. (1999). Mobile agent applications. *IEEE Concurrency*, pages 80 – 90.

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