Computers & Security Vol.20, No.6, pp.525-532, 2001
Copyright © 2001 Elsevier Science Limited
Printed in Great Britain. All rights reserved
0167-4048/01$20.00

Access Control in Document-centric Workflow Systems – An Agent-based Approach

Reinhardt A. Botha[1] and Jan H.P. Eloff[2]

[1] *Faculty of Computer Studies, Port Elizabeth Technikon, Port Elizabeth, South Africa, e-mail: rbotha@computer.org*
[2] *Department of Computer Science, Rand Afrikaans University, Johannesburg, South Africa, e-mail: eloff@rkw.rau.ac.za*

Workflow Systems are increasingly being used to streamline organizations' business processes. During the execution of business processes, information often traverses organizations' networks as documents. With the proliferation of the Internet, documents travel across open networks. These documents can, however, contain potentially sensitive information. The documents used in Workflow Systems must therefore be protected from unauthorized access.

This paper enumerates three access control requirements of workflow environments, including the well-known principle of separation of duty. Thereafter the CSAC (Context-sensitive Access Control) model is presented to address the requirements. In conclusion it is demonstrated how this model can be implemented in an agent-based architecture.

Introduction

Workflow Systems increasingly use the Internet as the underlying communications infrastructure. With the implementation of Workflow Systems, organizations target their core business processes in order to get a return on investment as fast as possible. These processes often deal with sensitive information. Consequently serious information security concerns are raised.

Similar to other systems, the information security requirements of Workflow Systems are modelled according to the five security services suggested by ISO 7498-2. [3][6] These five services are: identification and authentication, access control, confidentiality, integrity and non-repudiation. To secure a Workflow System effectively, mechanisms for each of these services must be employed.

Three of these services, namely identification and authentication, confidentiality and non-repudiation are implemented similarly in Workflow and non-Workflow Systems. However, the access control service and integrity service require special attention. This paper primarily addresses access control. However, it can be argued that the access control service assist with integrity in the sense that it prevents unauthorized users from changing the data.

In Workflow Systems, the access control requirements must be expressed based on the types of tasks that must be performed on the information objects that require protection. Documents are the most common way through which information in an organization is communicated. In order to develop an access control model for Workflow Systems, the access control requirements of the Workflow Systems need to be established.

Access Control Requirements of Workflow Systems

Various authors identified that Workflow Systems have special access control requirements. [1][2][5] For the purposes of this paper, the access control requirements of Workflow Systems are described through three properties.

> **Property 1: Strict least privilege.** The concept of least privilege acknowledges that users should only receive access permissions that are in line with their job responsibilities. It does, however, not recognize that those permissions may at specific times be inappropriate and unnecessary. For example, a manager who initializes a purchase order should not, at the initialization stage, receive the permission to approve the purchase order. At a later stage, however, the manager may indeed receive the permission to approve a purchase order. Strict least privilege involves the strengthening of the least privilege concept in that it distinguishes between a person's job and the tasks that a person must fulfil as part of his job. Strict least privilege therefore states that a user should receive the smallest possible set of permissions for the current task within the business process.
>
> **Property 2: Sequence of events.** Certain permissions can be granted only once others have been exercised. The event of granting a specific permission may thus depend on the completion of another task, i.e. the exercising of other permissions. For example, an order cannot be approved until filled out completely; similarly, once an order has been approved, it may not be re-edited.
>
> **Property 3: Separation of duty.** Separation of duty has as its primary objective the prevention of fraud and errors, thus ensuring the semantic integrity of business information. Separation of duty requirements are often formulated as business rules such as: "a person may not approve his own purchase order" or: "a cheque requires two different signatures." In this case, the access control service should be sensitive to the access history of the relevant objects and appropriately disallow access.

The remainder of the paper is structured as follows: An access control model that meets the three identified access control requirements, the CSAC (Context-sensitive Access Control) model, is developed in the next section. Thereafter, an agent-based architecture for the implementation of document-centric workflow is presented. Subsequently, it is explained how the access control model maps onto the architecture. The paper concludes by discussing practical issues when implementing the access control model in the agent-based workflow architecture.

CSAC: An Access Control Model for Workflow Systems

The model developed in this paper builds on well-accepted Role-based Access Control (RBAC) principles. Additional concepts that facilitate the needs of the workflow environment are introduced.

RBAC

The interpretation of RBAC hinges on the concept of a role. Permissions and users are both assigned to roles. A user receives the permissions associated with the role(s) that the user may assume. Formally, the following sets can be identified:

USER	the set of users who are capable of performing actions in the system;
ROLE	the set of all roles in the application;
OBJ	the set of all objects in the system;
METH	the set of actions that may be performed on objects in the system.

Because methods are dependent on the semantics of the objects in the system, permissions are defined as the methods that may be performed per object, i.e.

$$PERM \subseteq 2^{OBJ} \times 2^{METH}$$

Roles are related to each other according to a partial order. Roles inherit the permissions of the roles that are junior to them in the partial order.

$$RH = (ROLE, \leq)$$

The following function will return all the users that belong to a role or a role superior in the partial order to that role.

$$U: ROLE \rightarrow 2^{USER}$$

The concepts encountered in RBAC models do not fully match the concepts found in the workflow environment. Thus, additional concepts as encountered in Workflow Systems need to be introduced.

Workflow concepts

Workflow Systems refer to the software that facilitates the modelling and execution of business processes. The business process is described to the Workflow System by means of a process definition. The process definition is constructed during build time. A process definition identifies the tasks that form part of the business process, and provides business rules that specify the conditions for the execution of the tasks.

A workflow is executed in the run-time environment. During run-time, process instances are created based on the specification provided by the process definition. In a purchase order process, a process instance will be generated for each purchase order that is generated. Task instances will be created on demand in the run-time environment depending on the interpretation of the business rules that form part of the process definition.

Tasks can thus be identified as the principle building-blocks of Workflow Systems. A task can be performed by certain roles. The *TASK* set can thus be introduced to represent the various task definitions. The following function maps tasks to the roles that may perform them:

$$T: TASK \rightarrow 2^{ROLE}$$

A task requires certain permissions to be completed. The function:

$$P: TASK \rightarrow 2^{PERM} \qquad \text{(A)}$$

associates tasks with permissions. A Workflow is considered a partially ordered set of tasks:

$$W = (TASK, \leq)$$

If $t_1 \leq t_2$ then t_1 is executed in parallel with or before t_2. The discussed constructs are all set up during build time. The next subsection introduces concepts related to the run-time enforcement of access control.

Run-time enforcement: The WSession concept.

Users do not always need to fulfil all roles that have been associated with them. A WSession is used to control the activation of roles in the workflow environment based on the task instance that is currently being dealt with.

The need for strict least privilege (Property 1) in a workflow environment requires the user to assume the absolute minimum role required for the task. When a user is chosen for a task, it is done based on the roles that the user may assume. However, in order to support the concept of strict least privilege, a WSession associates a user with permissions based on the requirement of the task and not the user's role.

We can thus define *WSession* as follows:

$$WSession = \{(u, P(t)) \mid (\exists r \in T(t)): u \in U(r)\}$$

This also addresses the sequence of events requirement (Property 2). If permission pi must only be executed after permission pj, the administrators of the system must ensure that the following invariant is true.

$$P_i \in P(t_k) \wedge p_j \in P(t_m) \Leftrightarrow t_k \leq t_m \qquad \text{(Invariant 1)}$$

This section demonstrated how properties 1 and 2, viz. strict least privilege and sequence of events, could be attained. The next section extends these ideas to incorporate separation of duty (Property 3).

Separation of duty

As already stated, separation of duty has as its primary objective the prevention of fraud and errors. There are two strategies for the enforcement of separation of duty requirements. Certain SoD requirements can be enforced in the build-time environment, viz. static separation of duty. Static separation of duty requirements will therefore restrain the build-time associations between the entities. For example, the same user may never be an auditor and a financial manager. Alternatively, SoD requirements can be enforced in the run-time environment, viz. Dynamic separation of duty. Dynamic separation of duty requirements do not constrain the build-time associations between entities. They do, however, restrict how these associations are activated at run-time. For example, a user may not be restricted to belong to the manager role, but may not assume that role for a purchase order that he himself initiated.

Static separation of duty, being the responsibility of the administration and design tools, is not discussed in this paper. This paper concentrates only on dynamic separation of duty requirements.

The concept of conflicting roles has been identified as a means of expressing separation of duty requirements. [4] This paper shows how separation of duty (Property 3) can be implemented by using the concepts of conflicting tasks and conflicting users.

Conflicting users are defined as users who are likely to conspire. In practice, family members may be considered conflicting users. Conflicting tasks are tasks that should not be performed by the same user or by two conflicting users. In terms of the example, a typical SoD requirement is that a purchase order may not be approved by the originator or a family member of the originator of the purchase order.

This requirement has, therefore, an impact on how the users are selected. The selection of users would not only involve the roles that a user may fulfil, but must also consider the access history throughout the life of the document. For this reason, a baggage database, i.e. a history of relevant accesses to the document, will be maintained. We define baggage as a database consisting of tuples of the following format:

$$(n,t,u,r,m) \in N \text{ x } TASK \text{ x } USER \text{ x } ROLE \text{ x } METH$$

Since the baggage is stored for each document, the baggage database of a specific document is denoted by DB_o where $o \in$ OBJ. Not all the tuples in DB_o are meaningful. The following invariant must be maintained:

$$(n,t,u,r,m) \in DB_o \Leftrightarrow [r \in R(u)] \wedge [r \in T(t)] \wedge [(o,m) \in PERM] \qquad \text{(Invariant 2)}$$

The separation of duty constraints can be modelled as further invariants that must be maintained. To facilitate their specification, two functions to identify the conflicting tasks and conflicting users are specified as follows:

CT: $TASK \rightarrow 2^{TASK}$, and
CU: $USER \rightarrow 2^{USER}$

The set of conflicting tasks returned by CT will contain at least the task used as a parameter. Similarly, CU will return at least the user used as a parameter. In order to enforce separation of duty, the following invariant must furthermore be maintained: (The # indicates that the value does not matter.)

$$(n,t,u,r,p) \in DB_o \Leftrightarrow [\forall u_m \in CU(u)][\forall t_k \in CT(t)]: (n,t_k,u_m,\#,\#) \notin DB_o \qquad \text{(Invariant 3)}$$

This model showed how the three properties required for access control in Workflow Systems can be realized. Property 1, strict least privilege, was achieved by associating the function P (see (A)) with the WSession object. Property 2, order of events, will be achieved if Invariant 1 is maintained. Property 3, separation of duty, requires the maintenance of Invariants 2 and 3.

The paper now proposes an agent-based architecture for document Workflow Systems. Thereafter it will be shown how the CSAC model can be incorporated into the architecture in order to implement the three access control properties.

An Agent-based Architecture for Document-centric Workflow Systems

The architecture of the proposed agent-based Workflow System consists of the following five agent types:

User Agent. The User Agent is the interface between the user and the incoming work;
Profile Agent. The Profile Agent is responsible for the maintenance of organizational information. It is therefore responsible for maintaining a database of all users in the system, the roles in the

system, as well as the mapping between users and the roles they may assume;

Document Agent. A Document Agent is the custodian of a document. It is generated when the document is created. All access to the document occurs through the Document Agent. The Document Agent will see the lifecycle of a document as a state machine. It will note the state of the document and determine which roles may access the document based on the state. The Document Agent will record the access history for the document in a baggage database;

Session Agent. The Session Agent is responsible for coordinating the activities involved in a user's work session. It will coordinate the various Document Agents that may form part of a specific task. It acts as an intermediary between the Document Agents and a User Agent. A Session Agent is thus responsible for maintaining information about the specific access permissions granted to, and exercised by, the user performing the task;

Routing Agent. The Routing Agent is responsible for the routing of work. It controls the creation of Session Agents according to the business rules contained in the process definition.

Consider how a typical process instance is handled. Figure 1 graphically depicts the lifecycle of the agents throughout the execution of a process instance. Note that the persistent agents, i.e. those that exist for longer than the duration of the process instance, are indicated by solid lines, whereas the volatile agents are indicated by a broken line. The following stages can be identified:

(A) A Routing Agent is created for the new process instance;

(B) The Routing Agent creates a Session Agent who will be responsible for overseeing the execution of a task;

(C) The Session Agent determines which documents are required and requests them to migrate to the Profile Agent. If a new document must be

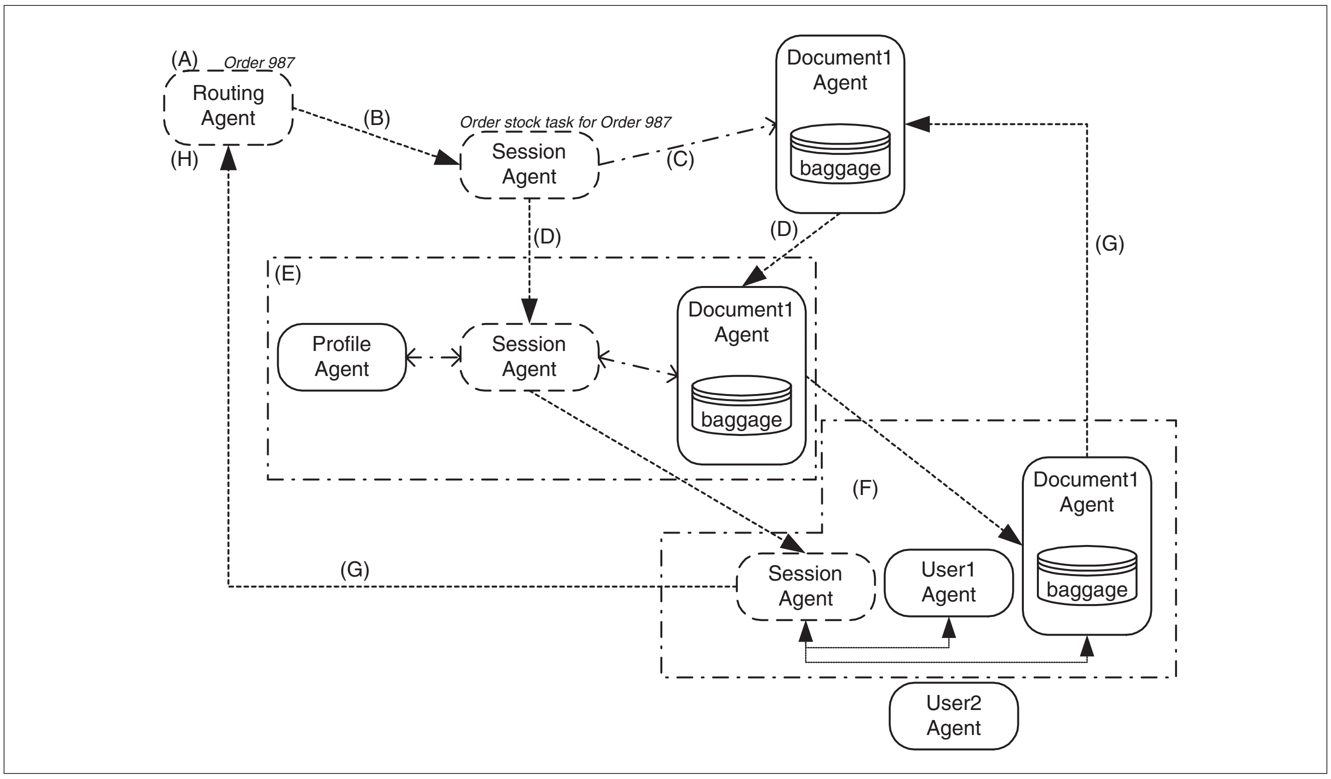

Figure 1: Lifecycle of agents throughout a process instance.

created, the Session Agent will create the relevant Document Agent;

(D) The Session Agent and the Document Agent migrate to the Profile Agent;

(E) The Profile Agent, the Session Agent and the Document Agents negotiate on possible users that may execute the task. The Document Agents interrogate their baggage databases to ensure that inappropriate users are not assigned to the task;

(F) After choosing a user, the Session Agent and the Document Agent migrate to the User Agent. The Document Agent will maintain access history as part of its baggage;

(G) Once the task has been completed, the Session Agent migrates back to the Routing Agent, and the Document Agents migrate to their persistent storage. The Session Agent is destroyed;

(H) The Routing Agent will determine the next task to be performed. Once all the tasks have been completed, the Routing Agent will be destroyed.

The next section shows how the CSAC model can be implemented in an agent-based environment that follows this architecture.

Implementing CSAC in an agent-based architecture

The discussion on implementing CSAC in the proposed agent-based architecture commences with an overview of the physical distribution of information in the agent-based architecture. Thereafter, the maintenance of the invariants is discussed in the context of the agent-based architecture.

Mapping CSAC components onto the agent-based architecture

Figure 2 graphically depicts the mapping of the CSAC components to the agents that were identified in the proposed agent-based architecture. Consider each of the agents in turn.

The ***Routing Agent*** will be generated containing the workflow definition *W*. As such, it contains information about the tasks and the information that it requires for making decisions about the flow between tasks. The Routing Agent will thus also contain the functions *P* and *T* that return the required set of permissions and the possible set of roles respectively. The Routing Agent will furthermore contain the function *CT* that identifies conflicting tasks.

When the ***Session Agent*** is created, it will contain the results of functions *P*, *T* and *CT* for the task for which the session is responsible. The Session Agent will furthermore contain the WSession object whose contents will be created when the Session Agent, the User Agent and the Document Agents collaborate to execute a task.

A ***Document Agent*** is created for each document in the system. The *OBJ* set thus represents all the Document Agents. A Document Agent encapsulates the actual document and the baggage database. The interface to the Document Agent represents the *PERM* set. The Document Agent will have knowledge about the lifecycle of a document and as such knows the state of the document. The Document Agent will contain the mapping of permissions allowed, based on the state of the document.

The ***Profile Agent*** contains information about the structure of the organization. It will thus contain the *USER* and *ROLE* sets. Furthermore, it will contain partial order *RH*, as well as the implementation of functions *U* and *CU*.

The ***User Agent*** only serves as an interface between the system and the user. As such, the collection of User Agents presents the set of users in the system.

The formal model developed in the previous section provided a set of invariants that must be maintained. It is important that these invariants should not only prevent the occurrence of an error once an undesired situation arise, but the execution of the workflow should also be planned to minimize the chances of an undesired situation arising. The following section provides implementation suggestions on the maintenance of the invariants in the model.

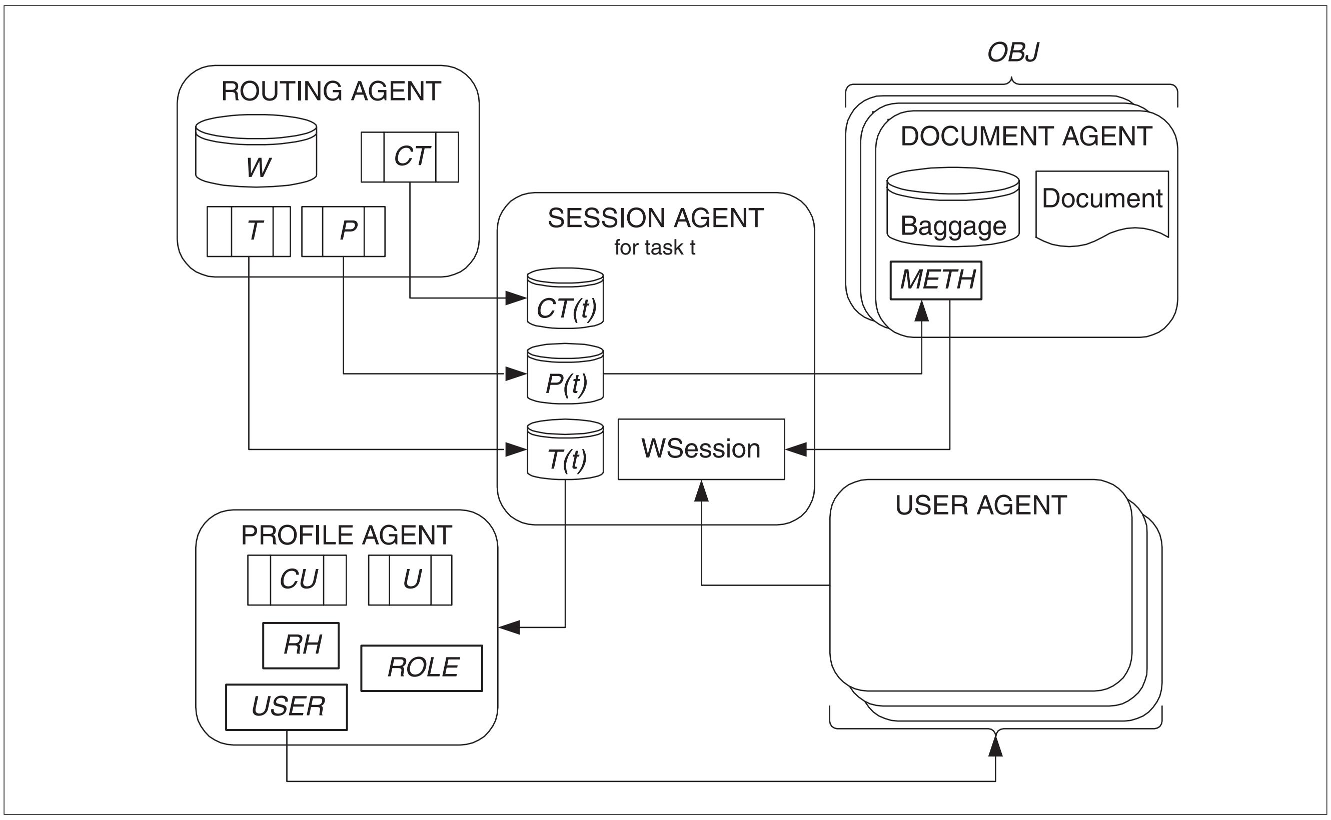

Figure 2: Mapping the CSAC model compontents to the agent-based architecture.

Implementing the CSAC invariants

The CSAC model provided a set of invariants that must be maintained.

Invariant 1 is concerned with the definition of the process. As such, invariant 1 must be maintained during build-time, thus falling outside the scope of the proposed agent-based architecture. Invariants 2 and 3 are based on values that should never exist in the baggage of a document. Baggage is written because of interaction with the Document Agent. Since the Session Agent is mediating all the communication with the Document Agents, the Session Agent will be responsible for ensuring that invariants 2 and 3 are maintained. Consider each of the invariants in turn.

Invariant 2 is concerned with the soundness of the tuples inserted into the database. It was given as:

$(n,t,u,r,m) \in \mathrm{DB}_o \Leftrightarrow [r \in R(u)] \wedge [r \in T(t)] \wedge [(o,m) \in PERM]$ (Invariant 2)

During the collaboration between the Profile Agent, the Session Agent and the relevant Document Agents, a user will be chosen only if the user can perform the role, i.e. if $r \in R(u)$. Furthermore, only roles that may perform the task will be considered, i.e. $r \in T(t)$. The results of function P for a specific task will be stored by the Session Agent. These results will return tuples of the format (o,m). Since this information is available to the Session Agent, which acts as an intermediary between the User Agent and the Document Agents, it is arbitrary to ensure that (#,#,#,#,m) only gets written to DB_o.

Invariant 3 is concerned with the implementation of property 3, separation of duty. It was given as:

$(n,t,u,r,p) \in \mathrm{DB}_o \Leftrightarrow$

$$[\forall u_m \in CU(u)][\forall t_k \in CT(t)]:(n,tk,um,\#,\#) \notin DB_o$$
(Invariant 3)

The problem with maintaining the invariant is that it implies that the check is done only when new tuples are inserted into the baggage database. New tuples are written away to the baggage database only once a method has been invoked by the Document Agent. By then, it is too late to prevent the action. The system should thus prevent the action from ever occurring. Thus, when the Profile Agent, the Session Agent and the Document Agent(s) collaborate (step (E) in Figure 1) to identify a user suitable to perform the task, unsuitable users should be disregarded. This can logically be implemented in the following way:

First, a list of users that, based on their assigned roles, may perform the task. This is given by $U_{original} = U(R(T(t)))$

The relevant Document Agents must then assemble a list of users based on the conflicting tasks that appear in their baggage. This list is described as:

$$U_{didconflictingtask} = \{u \mid [\exists (o,m) \in P(t)][\exists t_x \in CT(t)]:(n,tx,u,\#,m) \in DB_o\}$$

The list can then be expanded to include conflicting users:

$$U_{extended} = \{u_x \mid [\exists u \in U_{didconflictingtask}]:u_x \in CU(u)\}$$

The list of users can be calculated as $U_{original}$ minus $U_{extended}$. This list excludes all users who would cause invariant 3 not to hold . The Session Agent can thus safely select an appropriate user to perform the task from the list.

Conclusion

In this paper three requirements for access control in Workflow Systems were identified. An agent-based architecture for the implementation of document-centric workflows was proposed. A model, based on role-based access control principles, to address the three requirements of access control in Workflow Systems was developed. Thereafter, it was demonstrated how the model could be incorporated in the suggested agent-based architecture. Finally, the paper discussed practical implementation issues.

This paper showed that role-based access control principles are suitable for implementation as an access control mechanism in an agent-based workflow environment. The introduction of persistent Document Agents provides the freedom of documents that can roam an open environment in a secure fashion.

References

[1] Atluri, V. and Huang, W-K. (1996) "An Authorization Model for Workflows", Proceedings of the fifth European Symposium on Research in Computer Security, Rome, Italy, pp. 44 – 64.

[2] 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, Vol. 2, No. 1, pp 65 – 104.

[3] ISO (1989) ISO 7498-2: Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture

[4] Kuhn, D.R. (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, Fairfax, VA, pp. 23 – 30.

[5] Long, D.L., Baker, J. and Fung, F. (1999) "A Prototype Secure Workflow Server" Proceedings of the 15th Annual Computer Security Applications Conference, Radisson Resort Scottsdale, Phoenix, Arizona, pp. 129 – 138.

[6] Workflow Management Coalition. Workflow Security Considerations – White Paper. Document number WFMC-TC-1019 Issue 1.1 Available from www.wfmc.org