An Authorization and Access Control Model for Workflow

Sodki Chaari[1], Chokri Ben Amar[1], Frederique Biennier[2] , Joel Favrel[2]

[1] Ecole Nationale d'Ingénieurs de Sfax, TUNISIE
[2] Institut National des Sciences Appliquées de Lyon, FRANCE

Abstract. A workflow is a coordinated arrangement of related tasks in an automated process, the systematic execution of which, ultimately achieves some goal. Workflow Management Systems (WFMSs) are becoming very popular and are being used to support many of the day to day workflows in large organizations. Security is an essential and fundamental part of workflows, the WFMS has to manage and execute the workflows in a secure way. Security, in a workflow context, involves the implementation of access control security mechanisms to ensure that tasks are performed by authorized subjects only. In this paper we propose a workflow authorization model capable of specifying authorization in such a way that subjects gain access to required objects only during the execution of the task. We build our model over the well known RBAC framework, and that in addition extends RBAC by including new rules in order to be adaptable with workflow context.

1 Introduction

Workflow (or workflow process) is the computerised facilitation or automation of a business process involving the coordinated execution of multiple tasks performed by different processing entities. A workflow can be abstracted as a network with task (i.e., activity) nodes and flows (i.e., transitions between task nodes). A task defines a logical unit of work in a workflow that related to a specific commitment, adding value to a product or service of an organization. A workflow also defines task dependencies (transitions) that specify how tasks in a workflow are coordinated for execution in a semantically correct order [5]. Workflow Management Systems (WFMS) are used to coordinate and streamline business process in numerous applications domains including office automation, banking, healthcare, telecommunications and manufacturing. WFMS are used for critical and strategic applications, they often use heterogeneous and distributed hardware and software systems to execute a given workflow. This gives rise to decentralized security policies and mechanisms that need to be managed. Since security is an essential and fundamental part of workflows, the WFMS has to manage and execute the workflows in a secure way. The security service of authorization (access control) is of primary relevance in the context of workflows. Access control security mechanisms need to ensure that task are

Chaari S., Ben Amar C., Biennier F. and Favrel J. (2004).
An Authorization and Access Control Model for Workflow.
In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*, pages 31-40
DOI: 10.5220/0002663400310040

performed by authorized subjects only. An appropriate authorization model for workflows must provide the notion of just-in-time authorization. It enables the granting, usage tracking and revoking of authorization to be automated and coordinated with the progression of various tasks. Otherwise, a subject may process authorization for time periods longer than required, which may compromise security. The model has to prevent any unauthorized modification of data and to enforce the legitimate pattern of operations in data accesses by the subject(s) for executing a task.

In this paper, we propose an authorization and access control model that is able to specify authorization in such a way that subject gain access to required object during the execution of the task. We build our models over the well-known role-based access control (RBAC) framework. In our authorization model we try to inject RBAC into an existing workflow system. We extend the RBAC model by adding some new rules and definitions to meet our needs and to be able to deal with the workflow context. In the rest of the paper, section 2 describes some related work, section 3 describes the basic elements of the model, section 4 presents the global workflow authorization model and finally section 5 discusses the conclusions and some perspectives.

2 Related work

Security is a critical and essential part of workflows, and it has become an important topic in the research community as well as the industry especially authorization and access control have been widely discussed and many methods have been proposed to model the authorization and access control properties. Many researchers are working on workflow standards. The Workflow Management Coalition (WfMC) is a non profit organization that focuses on the advancement of workflow management technology in industry. WfMC summarizes a number of security services [9] for a conceptual workflow model that includes authentication, authorization, access control, data privacy, audit…

For workflow security, previous research has been done mainly on several aspects, which include task assignment constraints, inter-workflow security, and multilevel secure workflow systems [7]. Using task assignment constraints, assignment methods for the workflow systems are specified in terms of constraints on the permissible assignments of users to tasks and roles. Because the role-based model is a natural choice for implementing security in workflow systems, most of the discussions are based on that.

Bertino, Ferrari, and Atluri [2] propose an interesting and powerful constraint based security model also based on logic predicates, that allows for somewhat different expressivity than the one presented here. Predicates in constraint expressions include predicates over a role graph and predicates over history.

The Workflow Authorization Model (WAM) [1] presents a conceptual, logical and execution model which concentrates on the enforcement of authorization flow in task dependency and transaction processing by using Petri Nets (PN). The workflow designer defines the static parameters of the authorization using an Authorization Template (AT) during the build-time of the workflow. When the task starts execution, the AT is used to derive the actual authorization. In a Multilevel Secure (MLS) workflow environment, tasks are assigned to different security levels.

WAM discusses the synchronization of authorization flow with the workflow and specification of temporal constraints in a static approach, it is not sufficient to support workflow security. This is because workflows need a more dynamic approach to synchronize the flow of authorizations during the workflow execution. For example, the privileges will be granted/revoked to/from the users according to the events generated during the task execution. WAM only concentrates on the authorization in a task's state and primitives.

Previous studies focus either on Inter-workflow security is concerned with the security of the communication and cooperation of autonomous workflow systems, running at different units of the same organizations or at different organizations. Some related work can be found in [8].

3 Basic elements of the model

This section describes our basis elements for the access control that uses the roles like a means to classify several permissions under the same name. We give the precise definition of a role and of other entities that form the basis of a Role Based Access Control (RBAC) model that is presented in this work.

3.1 Permissions

The security of a computer system in an access control model generally accepts that permission describes an approval of a particular access right to an object.

Definition 1
A permission *pr* is a pair (tr, Obj) where *tr* is the transaction that operates on the set of objects *Obj*.
Let entities namely, sets of permissions (P), transactions (Tr) and objects (Obj) respectively, be: $P = \{pr_1, pr_2, ...\}$ the set of permissions, $Tr = \{tr_1, tr_2, ...\}$ the set of transactions and $Obj = \{obj_1, obj_2, ...\}$ is the set of objects .
The relationships among these entities are the following:

- $TrP : P \to Tr$, gives the transaction associated with the given permission.
- $ObjP : P \to 2^{Obj}$, gives the set of objects associated with the given permission.

3.2 Roles

A role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility [6]. A role is formed by the grouping of permission according to logical and functional needs.

Definition 2
A role *r* is a pair $(rname, rpset)$ where *rname* is the role name, and *rpset* is the set of role permissions.
Let $R = \{r_1, r_1, ...\}$ denote the set of roles we express the mapping between roles and permissions with: $PR : R \rightarrow 2^P$, gives the set of permissions for the given role r.

3.2.1 Role-user authorizations

The permissions associated with a role are administered as a single unit such that authorization to access a role puts all the role's permissions at the disposal of an authorised user and thus confers access rights grouped in the role to the user.

Definition 3
A user authorization list UAL is of the form $[u_1, u_1, \cdots u_n]$, where u_i is the user identifier. Let $U = \{r_1, r_1, ...\}$ denote the set of users identifiers then a user authorization list can also be defined as the mapping function between roles and users:

- $UR : R \rightarrow 2^U$, which enumerates the users associates with the given role r.

3.2.2 Role hierarchy

Tracking the distribution of such permissions among roles can become a very complex task. In order to simplify administration and analysis of permission distribution, it is important to have some means for organising the various roles. Such structures of roles, defined by [3] as role hierarchies, are often discussed alongside roles.

Definition 4
An inheritance relation $\rightarrow$ is defined between two roles r_i and r_j, denoted $r_i \rightarrow r_j$ if and only if $PR(r_i) \subseteq PR(r_j)$.

In this relation role r_j is seen as a senior role, whereas r_i is a junior role. In addition to directly assigned permissions role r_j indirectly contains the permissions of the junior role r_i. Thus, the function $PR(r)$ returns not only all direct permissions of the role, but also all indirectly contained (inherited) permissions.

3.2.3 Role graph

The hierarchies of roles with inheritance relations among them are combined to form a directed acyclic graph, such that with the roles as nodes for a given directed edge.

Definition 5
A role graph is an inheritance relation on the set of roles, defined as $RG = (R \ X \ R \rightarrow)$.
The nodes are the roles from the set R, and the edges are defined with $\rightarrow$ relation.

3.3 Authorization State

While assembly the described elements until now we can form an authorization state that can be generated in the organization, formed by the following elements:

$$\left(Users, roles, user \quad authorizat\,ion \quad list, RoleGraph\right)$$

Therefore if a user belongs to the list of the users allowed for the role, or he has a senior role according to the graph of the roles, then he can execute the permissions that constitute the role and manipulate the different accessible objects with these permissions. This authorization can be judged with static. The name static comes from the fact that these authorization do not depend on the execution of any tasks, once it is defined by the system administrator it cannot be modified later taking into account the evolution of the system. Seen the dynamic behaviour of the workflow, this type of authorization cannot be used in the model of security that must be proposed. Indeed, for the workflow, we must insure that the authorization is only guaranteed during the execution of a task and revoked at the end of.

The roles authorised to a user have to be activated, so that he can use the associated permissions. An active role set is associated with each user, defined as the following mapping function: $ARS : U \rightarrow 2^R$, which gives the set of roles active for user.

The activation of the roles adds a dynamic character to the above authorization state, because the user is obliged to activate the adequate role in order to use the permissions associated with this role and we can verify the nature of the role activated as well as the date of the activation. This date is going to be very useful since we will compare it with authorized time for the execution of a task in the workflow.

3.4 Access Granting Rules

It is necessary to make some extensions to respond to the context of the workflow. The rules listed below can be assimilated to a gateway between the static behaviour of the role based access control and the dynamic one of the workflow.

Firstly, a user has to activate one or more roles authorised to him.

Rule 1

A user can activate a role if and only if the user is authorised to this role:

$$\forall u \in U, \quad r \in R: \quad u \in UR(r) \Rightarrow can\_activate\,(u, r)$$

Just when the roles are activated, the user has the right to make all permissions that are associated with the activated role, including those that are inherited.

Rule 2

A user can exercise permission if and only if the permission is an effective permission in the activated roles:

$$\forall u \in U, \quad p \in P: \exists \; r \in ARS(u): p \in PR(r) \Rightarrow can\_exercise\,(u, p)$$

Since the permission is defined as the right to execute a transaction on one or several specific objects, the last rule can be rewritten like follows:

Rule 3
A user can execute a transaction on an object if and only if a role exists on the set of the activated roles, and he possesses the permission that allows the execution of the transaction on this object:

$$\forall u \in U,\ \ tr \in Tr,\ \ obj \in Obj : \exists\ r \in ARS(u) \wedge \exists\ \ p \in PR(r)\, tr \in TrP(p) \wedge obj \in ObjP(p)$$
$$\Rightarrow can\_execute(u, tr, obj)$$

4 Authorization model for workflow

The different elements presented in the previous section have been specified with respect to users, roles and permissions, and granting access rights to users. This section contains the global description of the authorization model of the workflow. We are going to model the previous elements so that we can use them in the workflow context and present the rest of element that constitute our model.

4.1 Workflow model

Formally, a workflow is represented as a partially ordered set of tasks that is coordinated by task dependencies [1]:

Definition 6
A workflow W can be defined as a directed graph whose nodes are the tasks in the workflow $W = \{tw_1, tw_2, \ldots tw_n\}$ and edges are the task dependencies $tw_i \xrightarrow{x} tw_j$, where, $tw_i, tw_j \in W$ and x denotes the type of dependency.
Each task tw_i can be defined as a partially ordered set or totally of transactions $Tr_i\_set = \{tr_1, tr_2, \ldots tr_n\}$ that involve manipulation of objects [4].
Let $Obj = \{obj_1, obj_2, \ldots\}$ the set of the objects manipulated by the tasks of the workflow and $ObjT = \{objt_1, objt_2, \ldots\}$ the set of objects types. We define the mapping function: $F : Obj \rightarrow ObjT$, which associate a type to the object.
We use O_{objt_i} to denote the set of objects of type $objt_i$

4.1.1 Workflow task

Definition 7
A workflow task is defined as: $\left(ens\_Tr_i, ObjT_{IN_i}, ObjT_{OUT_i}, plan\_exec, \left[T_{\inf_i}, T_{\sup_i} \right] \right)$
where:
Tr_i_set is the set of transactions to be performed in tw_i , $ObjT_{IN_i} \subseteq ObjT$ is the set of object types allowed as inputs, $ObjT_{OUT_i} \subseteq ObjT$ is the set of object types expected as outputs, $\left[T_{\inf_i}, T_{\sup_i} \right]$ is the time interval during which tw_i must be executed.

Accordingly, we introduce the following mapping functions:

- $TrW : W \rightarrow 2^{Tr}$, gives the transactions in the given task.
- $TypeW : W \rightarrow 2^{Objt}$, gives the set of object types that can be processed in the given task.

4.1.2 Task instance

A task is a logical and abstract unit that assembly the particularities of an activity to achieve. Really, at the run time of a workflow, tasks instances are generated for all tasks that constitute the workflow. A task instance is a representative of the task. Any task may have several instances.

Definition 8

We can define a task instance as follows: $\left(TRANS_i, IN_i, OUT_i, exec\_plan, \left[T_{s_i}, T_{f_i} \right] \right)$ where:

$TRANS_i$ is the set of the transactions accomplished during the execution of the task, IN_i is the set of input objects to tw_i such that $IN_i = \left\{ x \in O \,\middle|\, F(x) \in Obj_{IN_i} \right\}$, OUT_i the set of output objects from tw_i such that $OUT_i = \left\{ x \in O \,\middle|\, F(x) \in Obj_{OUT_i} \right\}$, $exec\_plan$ specifies the order of transactions execution in the task and $\left[T_{s_i}, T_{f_i} \right] \in \left[T_{\inf_i}, T_{\sup_i} \right]$ is the time interval during which tw_i has been executed.

We define a function: $WInst : W \rightarrow 2^{T-INST}$ that gives the set of task instances.

4.2 Authorization generator (AG)

The major problem of a workflow is that the times of the beginning and the end of execution of one task cannot be predefined, they depend on the evolution state of the workflow. To solve this problem, it's important that the authorization must be assigned or revoked by the task according to its constraints. Therefore the task must be able to know when the authorization can be affected and when it must be revoked. The AG is the mean that allowed us to concretize this idea. It equipped the tasks by a certain "kind of intelligence" that made them able to manage their own authorizations.

Definition 9:

Given a task tw_i , an authorization generator $AG(tw_i)$ is defined as follows:

$$\left(p_i, r_i, type(objt_i, \_), \left[T_{\inf_i}, T_{\sup_i} \right] \right)$$

where:

p_i denotes the permission of role r_i to be enabled for executing transaction in the task: $TrP(p_i) \in TrW(tw_i)$, $type(objt_i, \_)$ is the function that verifies if the object is of type

$objt_i$ (objects able to reach the task), and $\left[T_{\inf_i}, T_{\sup_i} \right]$ is the time interval during which the task must be executed.

The authorization generators are attached to the tasks in a workflow. A new authorization is generated on a specific object if the type of this object is the same that the one in the function type of the generator. A task can have more that one generator.

An (AG) gives reference as to what permissions of which roles can be used during execution of a task. When a task starts the permissions of roles defined in a generator are enabled for the objects of the indicated type. Users that can use these permissions are not defined in an (AG), since the user's access to permissions is controlled via roles. Users must activate necessary roles in order to execute transactions of a task.

4.3 Authorization state

Every time that a transaction is programmed by the planner of the task and if there is presence of an object then an authorization is generated by the (AG).

Definition 10
An authorization is represented by: $A = \left(p, r, obj, task\_ins, \left[T_b, T_e \right], executors \right)$

This authorization indicates that the permission p of the role r can be used with the object obj in the task instance task_ins during the interval of time $\left[T_b, T_e \right]$.The set of executors represent user that have already used the permission.

An Authorization permits to affect a certain permission belonging to some role to a user in order to use it in a task instance. Parameters of an authorization are derived from the task authorization generator.

Definition 11
Every generated authorization is added to an authorization base (AB): $AB = \{A_1, A_2, ...\}$ the set of the authorization generated by the (AG).
For each authorization we also define the following mapping functions:

- $WInst\_AB : WInst \rightarrow 2^{AB}$, gives the set of authorization generated for a task instance.
- $PAB(A): AB \rightarrow P$, gives the permission enabled with authorization A.
- $RAB(A): AB \rightarrow R$, gives role name for which the permission is enabled.
- $OAB(A): AB \rightarrow Obj$, gives the object on which the action authorized.
- $Executors\ (A): AB \rightarrow 2^U$, gives users that finally used the enabled permission.

We can formulate the authorization rule that is adapted from the *Authorization Derivation Rule for WAM* [1] that was briefly discussed earlier. This rule will specify the granting and revoking of authorizations.

Definition 12 [Authorization Derivation Rules]

Given an authorization generator: $GA(tw_i) = \left(p_i, r_i, type(objt_i, \_) , \left[T_{\inf_i}, T_{\sup_i} \right] \right)$

An authorization $A_i = \left(p_i, r_i, obj_i, \ task\_ins_i, \left[T_{b_i}, T_{e_i} \right] , exécutors_i \right)$ is derived as follows: Suppose that the object O arrives to a transaction, chosen by the planner of a task instance tw_i, to the instant T_{a_i} (arrival time)

Grant Rule:

if $(O \in O_{objt_i}$ ($type(objt_i, O)$ is true) and $T_{a_i} \leq T_{\sup_i})$ then

$obj_i \leftarrow O$; $P_i \leftarrow p_i(AG)$; $r_i \leftarrow r_i(GA)$; $T_{e_i} \leftarrow T_{\sup_i}$;

if $(T_{a_i} \leq T_{\inf_i})$ then $T_{b_i} \leftarrow T_{\inf_i}$;

else $T_{b_i} \leftarrow T_{a_i}$;

Revoke Rule: Suppose that the transaction ends to the instant T_{f_i} (final time)

if $(T_{f_i} \leq T_{\sup_i})$ then $T_{e_i} \leftarrow T_{f_i}$;

Some new properties and rule are added to this algorithm so that it can adjust with our security model. When an authorization is initially generated the set of executors is empty, but fills in whenever an enabled permission is being used.

Property 1

A permission enabled in an authorization token can be used only by a user that is authorised to the role defined in the token, or to the role senior to it.

$\forall A \in AB, \ \forall u \in Executors\ (A)$ then $u \in UR(RAB(A)) \vee \exists\ r \in R$ such that

$(RAB(A) \rightarrow r) \wedge u \in UR(r)$

Property 2

A permission is an enabled permission of role *r* on object *obj* if there exists a valid authorization token within some task instance that authorises the permission of the role to be used on the object.

$\forall r \in R, \ \forall p \in PR(r), \ \forall obj \in Obj(p)$ then $p \in Enabled\_PR(r, obj) \Rightarrow$

$\exists\ tw \in W, \ \exists\ task\_ins \in WInst(tw), \exists\ A \in Winst\_ABA(task\_ins)$ such that

$p \in PAB(A) \wedge r \in RAB(A) \wedge obj \in OAB(A)$

Rule 4

A user can use a permission of the user's active role on the specific object only if the permission is enabled for this role.

$\forall u \in U, \ \forall r \in ARS, \ \forall obj \in Obj \ p \in Enabled\_PR(r, obj) \Rightarrow can\_exercise(u, p, obj)$

5 Conclusion

In this paper, we develop an authorization and access control mechanism to support workflow applications that is capable of synchronizing the authorization flow with the workflow. Several important aspects, such as task, role, permission, object, authorization generator, authorization base and authorization were introduced and considered in the model presented. We defined a model that mixes concepts of RBAC and Workflow with a different expressive power than previous models, which allow simpler specification of real life business processes.

The work presented in this paper can be extended along several directions. First, future work should be focused the proving of integrity and completeness of the model and on the improvement of the model. Second, incorporating issues related to the access control requirements for inter-organizational workflow into these models is also a challenging research goal.

References

1. Atluri, V., Huang, W.-K., Bertino, E.: An Execution Model for Multilevel Secure Workflows, in Proceedings of the 11th IFIP Working Conference on Database Security (1997) 151-165
2. Bertino, E., Ferrari, E., Atluri, V.: An Approach for the Specification and Enforcement of Authorization Constraints in Workflow Management Systems, ACM Transactions on Information Systems Security, Vol 1, No 1 (1999)
3. Ferraiolo David, F., Richard Kuhn, D., Cugini, J.: A. Role Based Access Control: Features and Motivations In Proceedings of Computer Security Applications Conference (1995)
4. Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management: From Process Modelling to Workflow Automation Infrastructure. Distributed and Parallel Databases (1995) 119 –153
5. Krishnakumar, N., Sheth, A.: Managing Heterogeneous Multi-system Tasks to Support Enterprise-wide Operations. The Journal on Distributed and Parallel Database Systems, 3 (2) (1995)
6. Sandhu, R.: Engineering Authority and Trust in Cyberspace: The OM-AM and RBAC Way. Fifth ACM Workshop on RBAC (2000) 111 – 119
7. Shengli, W., Zongwei L.: Authorization and Access Control of Application Data in Workflow Systems, Journal of Intelligent Information Systems, 18 (1), Kluwer Academic Publishers (2002) 71-94
8. Valia, R., Al-Salqan, Y.: Secure workflow environment. In: Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (1998) 269 – 276
9. Workflow Management Coalition (WfMC): Workflow Security Considerations White Paper, Document Number WFMC-TC- 1019, Document Status - Issue 1.0 (2001)