SOWAC: A Service-Oriented Workflow Access Control Model

XU Wei, WEI Jun, LIU Yu, LI Jing
*Technology Center of Software Engineering, Institute of Software,
the Chinese Academy of Sciences, Beijing 100080, P.R. China
E-mail: {xuwei, wj, liuyu, lij}@otcaix.iscas.ac.cn*

Abstract

Workflow access control is the fundamental issue in workflow security. With the development of enterprise globalization and the constant re-engineering and optimizing of enterprise business, the organization becomes more dynamic and its business process is frequently changing. As a result, workflow access control turns more complicated and entails a comparatively operational mechanism. To solve the problem, in view of decoupling workflow access control model from workflow model, we propose a Service-Oriented Workflow Access Control (SOWAC) model in this paper. In SOWAC model, service is the abstraction of a task and the unit for applying access control. We present the elements of SOWAC model and illustrate the enforcement of SOWAC with an example workflow. Then the dynamic separation of duty for SOWAC model is proposed based on the authorization history of services. By applying SOWAC in a real workflow management system, we show SOWAC model is practical and effectual.

1. Introduction

Workflow management system provides the infrastructure for Business Process Automation (BPA). With different organizations participating in workflow applications and each of them having its unique organizational structure, software systems and security requirements, workflow management system should support a process-level security mechanism. Workflow access control is the fundamental issue in workflow security [1]. The traditional access control model expresses an authorization as a tuple (s,o,p), specifying a subject (s) can gain privilege (p) to an object (o). To ensure that authorized subjects gain access on the required objects only during the execution of the specific task in a workflow application, granting and revoking of privileges need to be synchronized with the progression of the workflow from one task to another. This is not feasible unless there exists a workflow access control mechanism that authorizes an individual in synchronization with the progression of workflow.

Workflow access control mechanism aims to implement the security policies of an organization. First, it should fulfill the security requirements of an organization; second, it should realize the security goals of an organization. With the progression of enterprise globalization and the constantly re-engineering and optimizing of enterprise business, the organization becomes more dynamic and its business process is frequently changing. It necessitates re-assigning privileges to participants in business processes, thus increasing complexity of workflow access control. To deal with this, workflow access control model should be decoupled from the workflow model so that we are able to tune the workflow access control more dexterously to meet changes in workflows and organizations. In this paper, we propose a Service-Oriented Workflow Access Control model, SOWAC for short. In SOWAC model, service has been defined as abstraction of a task in a workflow, which describes interactions between workflow and its participants (users or applications). Therefore, SOWAC replaces access control on tasks with access control on services.

The remainder of this paper is organized as follows. In section 2, we investigate two major researches, RBAC and TBAC, on workflow access control. In section3, we present our SOWAC model, with its formalized description, enforcement and dynamic separation of duty that is based on the authorization history of services. Section 4 provides the application of SOWAC in ONCEPI, a practical workflow management system. Section 5 provides conclusions and future research directions.

2. Related Research

2.1. Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) has become a very popular access control paradigm. The best known of the RBAC models is the RBAC96 family of models proposed by Sandhu et al [2]. Figure 1 shows the main entities and their relationship with each other within the RBAC96 model. It can be seen that users are associated with roles. In turn, roles are associated with permissions. Users are typically humans or computer programs. Roles are defined in terms of responsibility that are often associated with a specific job function. As such, users are associated with roles based on their qualification to accept the responsibilities. Constraints (e.g., separation of duty) can apply to relations and functions in RBAC model. This is an effective mechanism for establishing higher-level organizational policy.

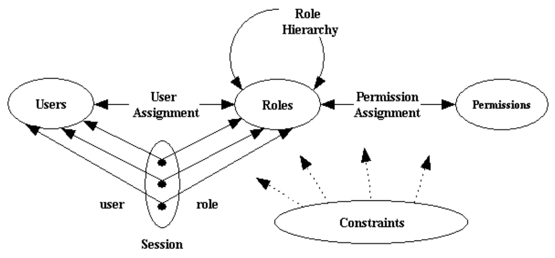


Figure 1. **RBAC96 model**

RBAC allows the specification of access control policy in a way that maps naturally to an organization's structure and the concept of a role is correspondent to an organizational position. So many researchers chose RBAC as the basis for workflow access control. Kandala and Sandhu extended RBAC96 to a secure workflow model by introducing explicit permission assignment (permissions assigned to a role) and implicit permission assignment (permissions on task instances) [3]. The permissions on task instances are assigned to roles based on explicit permission assignment. If a task is assigned to a role then all instances of the task are also assigned to the same role. Bertino et al. proposed workflow role specification that defined the role order for executing tasks. Furthermore, they present a constraint specification language to express the authorization rules [4]. Similarly, by associating tasks with roles, Miller et al. presented a workflow security framework using RBAC in the METEOR workflow project [5]. However, because RBAC does not consider workflow, all these researches focused only on associating tasks with roles, which leads to workflow model being tightly bound with RBAC model. Consequently, it is very difficult to re-configure workflow access control when workflow model or organization model has been changed.

2.2. Task-Based Authorization Control (TBAC)

Thomas and Sandhu recognized that with the increased automation of business activities there is a need for re-thinking the access control paradigms and they presented Task-based Authorization Controls (TBAC)[6,7]. TBAC is a task-oriented model for access control and authorization. It is an active security model that is well suited for workflow management system. From their task-oriented point, a high level task consists of several subtasks with multiple dependencies between the tasks that determine whether a particular permission may indeed be granted. It can be seen that this resembles the idea of a business process consisting of a network of tasks that are linked according to business rules. In TBAC, an authorization-step is used to describe the properties of a task and form the basis of modeling access control. Every authorization-step maintains its own protection state. The initial value of a protection state is the set of permissions that are turned on (active) as a result of the authorization-step becoming valid. However, the contents of this set will keep changing as an authorization-step is processed and the relevant permissions are consumed. Conceptually, permissions are checked-in and checked-out in a just-in-time fashion based on tasks. The TBAC family of models is, however, not defined to be orthogonal to workflow. Several workflow-related concepts are introduced as an intrinsic part of the models, so TBAC is coupled with workflow model tightly. In addition, TBAC does not provide tools for mapping the access control mechanism to its corresponding organization model. Therefore, it is too complex to implement TBAC in a practical workflow management system.

3. A Service-Oriented Workflow Access Control Model (SOWAC)

3.1. SOWAC Model

By decoupling access control model from workflow model, we propose a Service-Oriented Workflow Access Control Model (SOWAC). SOWAC also uses a role as the semantic construct forming the basis for access control policy, because the concept of a role facilitates modeling organizational hierarchy. However, rather than associates roles with tasks directly, SOWAC binds them through such a service as an interface. Service is an abstract expression of a task and permissions are assigned to services, instead of roles. Dependences between tasks, which are fundamental elements of TBAC, are counterparts of constraints for services in SOWAC. Figure 2 presents the SOWAC model and its relation with the elements of workflow. For the sake of brevity, a set of notations has been adopted that runs as follows:

- Wf: workflow
- WfIns: workflow instance
- T: task
- TIns: task instance
- U: a set of users
- R: a set of roles
- S: a set of services
- P: a set of permissions
- SS: a set of S-sessions

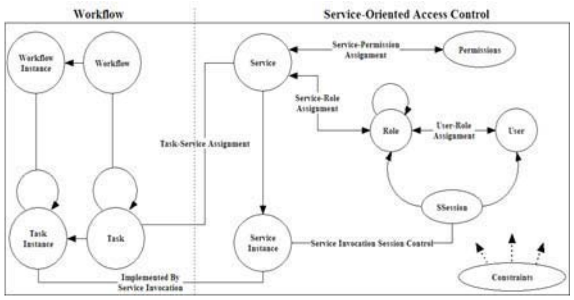


Figure 2. **SOWAC model and its relations with elements of workflow**

- C: a set of constraints

The elements, relations and functions of SOWAC are defined as below:

(1) A workflow (Wf) is represented as a partially ordered set of tasks and every task (T) is a 3-tuple $T = \{OP, D_{in}, D_{out}\}$. D_{in} is the set of input data types, D_{out} is the set of output data types, and OP is the set of operations that should be executed on T.

(2) $TS \subseteq T \times S$, a many-to-one task to service assignment relation. Each task has only one service as its abstract expression and one service can be assigned to several tasks.

(3) A workflow instance (WfIns) is a set of task instances $WfIns = \{TIns_1, TIns_2, ..., TIns_n\}$. Each TIns is corresponds to a service instance.

(4) $RH \subseteq R \times R$, a partial order on R that is called role hierarchy. This relation can be written as $\prec$. For example, $r_1 \prec r_2$ indicates that r_2 is the dominant role of r_1.

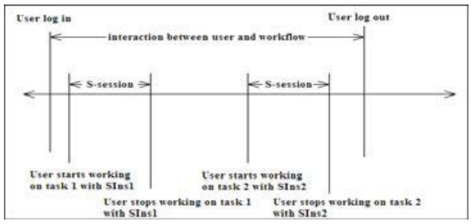
(5) $UR \subseteq U \times R$, a many-to-many user to role assignment relation.

(6) $SR \subseteq S \times R$, a many-to-many service to role assignment relation. If a service is assigned to the role r_i, this service is implicitly assigned to the dominant roles of r_i, $(\forall r_i, r_j : R)(\forall s : S) r_i \prec r_j \wedge s \in SR[r_i] \Rightarrow s \in SR[r_j]$.

(7) $SP \subseteq S \times P$, a one-to-many service to permission assignment relation. In the P set, permission is the abstract description of privileges for finishing a task. The nature of permissions is highly dependent upon the implementation details of the system, so we interpret the permissions for a workflow in terms of its components such as tasks and operations on them like execute.

(8) S is the set of services and we denote a service as $s = \{PA_{in}, PA_{out}, SP, \{SR, \prec_s\}\}$. PA_{in} and PA_{out} are, respectively, the set of input data types and the set of output data types. SP is the set of permissions assigned to s and SR is the set of roles assigned to s. The partial local order for SR is denoted $\prec_s$. There are three rules circumscribing activation of a role for a service s. First, if r_1 and r_2 are both assigned to s and $r_1 \prec_s r_2$, r_1 should be activated before r_2. Second, if r_1 and r_2 are both assigned to s, there is no partial local order between them and $r_1 \prec r_2$, r_1 should be activated before r_2. Third, if r_1 and r_2 are both assigned to s and they have no partial order relation in RH and SR, either of them can be activated.

(9) Service instance is a process of service invocation and we denote a service instance as SIns. SIns is a 4-tuple $SIns = \{d_{in}, d_{out}, ExeP, r\}$. In SIns, d_{in} is the set of input parameters and d_{out} is the set of output parameters. ExeP is the set of activated permissions for a service instance and these permissions are granted to r that is the activated role.



User log in
User log out
interaction between user and workflow
S-session
S-session
User starts working on task 1 with SIns1
User starts working on task 2 with SIns2
User stops working on task 1 with SIns1
User stops working on task 2 with SIns2

Figure 3. **S-session, user and service instance**

(10) SS is the set of S-sessions for invocating services and each S-session is used for one and only one service instance. During a S-session, a user is associated with a service instance, as shown in figure 3. Upon authentication to workflow management system, a user receives no permissions. When he embarks on a task, a service instance is thus created for the service assigned to this particular task. Also, a S-session is established for the user and the appropriate role and permissions of the service instance become available to the user. As soon as the user ceases working (suspend, cancel or finished) on the task, the S-session is closed and the service instance is destroyed. As a result, the corresponding role and permissions are revoked. There are two functions for a S-session: $suser: S-session \rightarrow User$, which maps a specific user to a S-session, and $ssi: S-session \rightarrow SIns$, which maps a service instance to a S-session. In run-time, SOWAC is able to support the principle of least privilege [8] through S-session.

(11) C is the set of authorization constraints enforced on different elements and relations of SOWAC model. Using a set of constraints is a powerful mechanism for laying out high-level organizational policy, such as separation of duty. The details of constraints will be discussed in section 3.3.

3.2. An Example

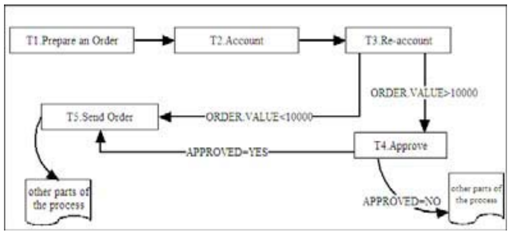


Figure 4. **An example workflow**

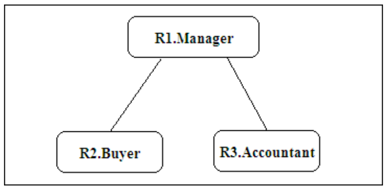


Figure 5. **Role hierarchy example**

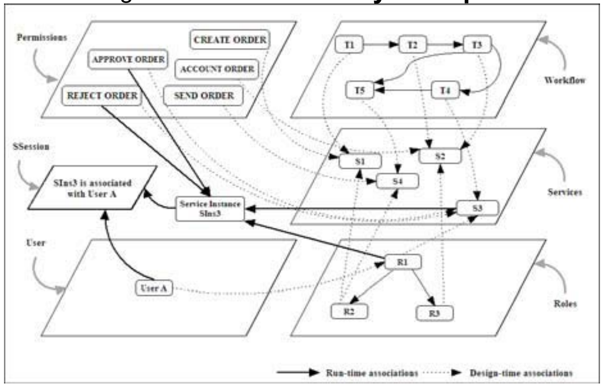


Figure 6. **Enforcement of SOWAC in the example workflow**

To illuminate the enforcement of SOWAC, we present an example workflow for SOWAC in this section. Figure 4 describes a portion of a typical purchasing workflow. In the portrayed workflow, a buyer will complete a purchase order (T1). Thereafter an accountant will account the order (T2) and another accountant will re-account the order (T3). Orders in excess of 10000 are forwarded to the manager for approval (T4), whereas the buyer send smaller orders to suppliers (T5). Figure 5 depicts the role hierarchy relating to the example workflow. As we can see, R1 is the role of manager, R2 is the role of buyer and R3 is the role of accountant. We define the set of permission P={CREATE ORDER, ACCOUNT ORDER, APPROVE ORDER, REJECT ORDER, SEND ORDER}. According to the requirements of data and operations in each task, we can define services as below.

(1) Service S1 for creating order task T1: S1={NULL, ORDER, {CREATE ORDER}, {{R1,R2},{ $R2 \prec_s R1$ }}}.

(2) Service S2 for accounting order task T2 and re-accounting order task T3: S2={ORDER, ORDER, {ACCOUNT ORDER}, {{R3, R1}, { $R3 \prec_s R1$ }}}.

(3) Service S3 for approving order task T4: S3={ORDER, APPROVED, {APPROVE ORDER}, {{R1},{}}}.

(4) Service S4 for sending order task T5: S4={ORDER, NULL, {SEND ORDER}, {{R2, R1}, { $R2 \prec_s R1$ }}}.

Figure 6 shows the associations between SOWAC elements of the example workflow. During the designing stage, three works need to be performed consecutively: (1) Generating the organization model through users, roles and role hierarchies; (2) Creating a set of permissions specific to the workflow; (3) Defining services and associating services with pertinent roles and permission. In the run-time, a service instance is created and assigned to an appropriate user through a S-session, e.g., in figure 6, user A acquired SIns3 during a S-session.

3.3. Dynamic Separation of Duty

With regard to the constraints in SOWAC, we focus only on separation of duty (SoD), because it is a well-known security policy and an important constraint in workflow security [1,4]. The purpose of SoD is to prevent fraud by requiring the involvement of more than one individual in completing a process [9,10,11]. In a workflow context, SoD has to be divided and extended into static and dynamic SoD [12]. Static SoD enforces certain rules during build time of the workflow and is therefore applied to the workflow specification. Static SoD is too strict to apply in a workflow management system. In contrast, dynamic SoD is enforced during run time, which is more flexible and adjustable. In this paper we will concentrate on dynamic SoD. The dynamic SoD for SOWAC is based on the authorization history of services. With the progression of a workflow instance, the authorization history of services is kept as the basis for enforcing next dynamic SoD constraints.

Perelson et al. proposed "conflicting entities" for SoD in workflow environment [13]. Conflict between entities, in the general sense, implies that the risk of fraud increases if associations with those entities are not carefully controlled. We apply the concept "conflicting entities" in SOWAC and present the definitions of four "conflict entities" with relevant examples from previous example workflow of section 3.2.

Definition 1: Conflicting permissions

Conflicting permissions are permissions that can result in unnecessary power if bestowed on the same person. $CP \subseteq P \times P$ represents the set of conflicting permissions, and $CPWith(p) = \{p_i \mid (p, p_i) \in CP\}$ is the function to identify the conflicting permissions of a specified permission p.

Example: "CREATE ORDER" and "APPROVE ORDER" are conflicting permissions.

Definition 2: Conflicting roles

Conflicting roles are roles that together possess the ability to conspire. $CR \subseteq R \times R$ represents the set of conflicting roles, and $CRWith(r) = \{r_i \mid (r, r_i) \in CR\}$ is the function to identify the conflicting roles of a specified role r.

Example: "Manager" and "Buyer" are conflicting roles.

Definition 3: Conflicting users

Conflicting users are users who will together have sufficient power to collude, and are likely to do so. In practice, this may be family members or previously known accomplices. $CU \subseteq U \times U$ represents the set of conflicting users, and $CUWith(u) = \{u_i \mid (u, u_i) \in CU\}$ is the function to identify the conflicting users of a specified user u.

Example: If use A and user B are brothers, they are conflicting users and should be treated as one user in a workflow instance. E.g., if user A has the permission "CREATE ORDER" in a workflow instance, user B will never be granted with the permission "APPROVE ORDER" in the same instance.

Definition 4: Conflicting services

Conflicting services are services that should be associated with conflicting permissions. $CS \subseteq S \times S$ represents the set of conflicting services, and $CSWith(s) = (s_i \mid (s, s_i) \in CS\}$ is the function to identify conflicting services of a specified service s.

Example: We can identify conflicting services by extracting mutual exclusive relations between tasks. Service S1 (for creating order task) and service S3 (for approving order task) are conflicting services.

The authorization history of services is denoted as SAH, which is the authorization base of dynamic SoD in SOWAC. SAH is used to track the authorizating process of service instances in a workflow instance. Each record in SAH, as the result of a S-session, is a 4-tuple $(sid, UsedP, r, u)$. For a S-session, "sid" is the identifier of a service that the service instance belongs to, "UsedP" is the set of permissions consumed by the user, "r" is the activated role for the user and "u" is the identifier of the user.

The following rules define the dynamic SoD constraints for SOWAC.

Rule 1:

In a workflow instance, conflicting roles should not be activated for the same user or conflicting users.

$$(r_i, r_j) \in CR \wedge (sid_i, UsedP_i, r_i, u_i) \in SAH \wedge (sid_j, UsedP_j, r_j, u_j) \in SAH \Rightarrow u_i \neq u_j \vee (u_i, u_j) \notin CU$$

Rule 2:

In a workflow instance, conflicting permissions should not be granted to the same user or conflicting users.

$$(p_i, p_j) \in CP \wedge (sid_i, UsedP_i, r_i, u_i) \in SAH \wedge p_i \in UsedP_i \wedge (sid_j, UsedP_j, r_j, u_j) \in SAH \wedge p_j \in UsedP_j \Rightarrow u_i \neq u_j \vee (u_i, u_j) \notin CU$$

Rule 3:

In a workflow instance, conflicting services should not be assigned to the same user or conflicting users.

$$(s_i, s_j) \in CS \wedge (sid_i, UsedP_i, r_i, u_i) \in SAH \wedge (sid_j, UsedP_j, r_j, u_j) \in SAH \Rightarrow u_i \neq u_j \vee (u_i, u_j) \notin CU$$

When enforcing dynamic SoD in a workflow instance, the S-session should execute these three rules by referring to the SAH and the sets of conflicting entities. Also, the record of current S-session will be added to the SAH. The result of calculating the dynamic SoD constraints can be presented by a set of users, denoted as DeniedU. Users in DeniedU are denied to access the service instance in the S-session. The following algorithm is used by the S-session to find out the DeniedU for a service instance.

Algorithm: dynamic SoD constraints based on SAH

Input: CP, CR, CU, CS, SAH, n (sid of the service), r, ExeP

Output: DeniedU

Steps:

Step 1: Calculating DeniedU$_1$, the set of users whose access are denied in accordance with rule 1, by the following function.

$$DeniedU_1 = \bigcup_{\forall r_i \in CRWith(r)} \{u \mid (sid_x, UsedP_y, r_i, u_z) \in SAH \wedge (u_z \in CUWith(u) \vee u_z = u\}$$

Step 2: Calculating DeniedU$_2$, the set of users whose access are denied in accordance with rule 2, by the following function.

$$DeniedU_2 = \bigcup_{\forall p_i \in CPWith(p) \wedge p \in ExeP} \{u \mid (sid_x, UsedP_i, r_y, u_z) \in SAH \wedge p_i \in UsedP_i \wedge (u_z \in CUWith(u) \vee u_z = u\}$$

Step 3: Calculating DeniedU$_3$, the set of users whose access are denied based on rule 3, by the following function.

$$DeniedU_3 = \bigcup_{\forall s_i \in CSWith(s_n)} \{u \mid (sid_i, UsedP_x, r_y, u_z) \in SAH \wedge (u_z \in CUWith(u) \vee u_z = u\}$$

Step 4:

$$DeniedU = DeniedU_1 \bigcup DeniedU_2 \bigcup DeniedU_3$$

4. Applying SOWAC in ONCEPI Workflow Management System

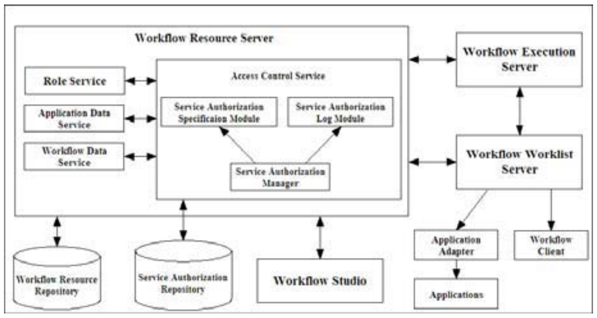


Figure 7. **Architecture of ONCEPI**

We apply SOWAC in a practical workflow management system, called ONCEPI, which is a process integration platform developed by TCSE of ISCAS. As shown in figure 7, ONCEPI is comprised of four major components: (1) Workflow Execution Server (WES) that is responsible for interpreting the specification of a workflow, creating and managing workflow instances, (2)Workflow Worklist Server (WWS) that manages the interactions with the workflow participants, (3)Workflow Resource Server (WRS) that provides services of workflow-related resource including role service, workflow data service, application data service and access control service, (4) Workflow Studio (WS) that is a unified workflow development environment and provides several tools, such as workflow modeling tool, organization modeling tool and access control configuration tool.

Access control service consists of an Authorization Specification Module, a Service Authorization Log Module, and a Service Authorization Manager.

Authorization Specification Module (ASM): This module provides interfaces for access control configuring tool to execute three types operations: defining services and permissions, associating services with roles and permissions, building the sets of conflicting services and conflicting permissions. All the result of these operations are stored in the Service Authorization Repository (SAR). In addition, other organization-related elements of SOWAC, i.e. roles, users, assignment relations between them, conflicting roles and conflicting users, are defined by organization modeling tool through role service of WRS and deposited in the Workflow Resource Repository (WRR).

Service Authorization Log Module (SALM): This module is responsible for managing the authorization log of services. When WES notified SALM that a new workflow instance is started, SALM will initialize a new SAH for this workflow instance and maintaining the SAH in synchronization with the progression of the workflow instance.

Service Authorization Manager (SAM): This module is to manage the service authorization through the S-session mechanism. Practically, it is the implementation of core functions of SOWAC. Those functions are creating and revoking of service instances, creating and revoking of the S-session, and enforcing the dynamic SoD constraints based on SAH.

During the execution of a workflow instance, WES sends an work item, which is the representation of work to be processed in the context of a task, to WWS, and then WWS will request a service instance from access control service by submitting the service ID and the workflow instance ID to WRS. Afterwards, WRS performs an authorization process to determine an executable service instance. The details of this process are presented as below.

(1) After receiving the service ID and the workflow instance ID, SAM queries the corresponding service definition and SAH from SAR by using functions of ASM and SALM.

(2) SAM creates a service instance based on the service definition and creates a S-session for this service instance.

(3) S-session calculates the DeniedU by executing the algorithm of the dynamic SoD constraints based on SAH.

(4) SAM sends the DeniedU and the activated role ID of the service instance to WWS.

(5) Using the data from SAM, WWS chooses a feasible user and sends the user ID to SAM.

(6) SAM allows the S-session to associate the user ID with the service instance, and then the S-session sends the service instance to WWS.

(7) WWS treats the service instance as the security context, including the set of executable permissions and the executable role, for the work item, and adds the work item into corresponding user's work list.

(8) After work item is finished, WWS returns the service instance to SAM.

(9) SAM revokes the S-session and asks SALM to add the recorder of current service authorization in SAR.

5. Conclusions and Future Work

In view of decoupling workflow access control model from workflow model, we have presented a Service-Oriented Workflow Access Control model, called SOWAC, in this paper. The novel part of SOWAC is the key concept of using a service as an abstraction of a task in SOWAC; thus traditional access control on tasks is replaced with access control on services. Hence, workflow model and organization model can be incorporated in SOWAC model, enabling resilience to possible alterations. Moreover, we have discussed the dynamic separation of duty based on the authorization history of services and presented a constraint algorithm. By applying SOWAC model, we have implemented a flexible access control in ONCEPI, a workflow management system. Currently, it is that we can only intuitively use definition of service in a straightforward way without taking into account detailed dependencies between tasks and even a precise method for extracting service from tasks and intricate relations in between. Our future work lies in coming up with some practical approaches to define services, thereby implementing a better service modeling in ONCEPI.

Acknowledgement

This research is supported by the National Natural Science Foundation of China (60173023, 60203029), the Chinese National "863" High-Tech Program (2001AA113010, 2001AA414020, 2001AA414330), and the Chinese National "973" Key Research Program (2002CB312005).

Reference

[1] Workflow Management Coalition, "Workflow Security Considerations – White Paper," Technical Report WFMC-TC-1019, Workflow Management Coalition, 1998. http://www.wfmc.org

[2] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, "Role-Based Access Control Models", *IEEE Computer*, 29(2): 38-47, 1996.

[3] S. Kandala, R.S. Sandhu, "Secure Role-Based Workflow Models", *Proceedings of IFIP TC11/WG11.3 Fifteenth Annual Working Conference on Database and Application Security*, Kluwer Press, July 2001, pp.45-58.

[4] E. Bertino, E. Ferrari, V. Atluri, "Specification and Enforcement of Authorization Constraints in Workflow Management Systems", *ACM Transactions on Information and System Security*, 2(1): 65-104, 1999.

[5] J. Miller, M. Fan, S. Wu, I. B. Arpinar, A. Sheth, K. Kochut, "Security for the METEOR Workflow Management System", Technical Report UGA-CS-LSDIS-TR-99-010, University of Georgia, June 1999. http://webster.cs.uga.edu/~budak/papers/security_report.pdf

[6] R.K. Thomas, R.S. Sandhu, "Conceptual Foundations for a Model of Task-Based Authorizations", *Proceedings of the Computer Security Foundations Workshop VII*, IEEE Press, June 1994, pp.66-79.

[7] R.K. Thomas, R.S. Sandhu, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management", *Proceedings of the IFIP TC11/WG11.3 Eleventh International Conference on Database Securty XI: Status and Prospects*, Chapman & Hall Press, August 1997, pp.166-181.

[8] K. Karlapalem, J. Gray. III., P.C.K Hung, "Issues in Document Security Enforcement for Activity Execution in CapBasED-AMS", *Proceedings of 12th International Conference on Information Networking*, IEEE Press, January 1998, pp.96-99.

[9] R.S. Sandhu, "Transaction Control Expressions for Separation of Duties", *Proceedings of the 4th Aerospace Computer Security Conference*, IEEE Press, December 1988, pp.282-286.

[10] R.S. Sandhu, "Separation of Duties in Computerized Information Systems", *Proceedings of IFIP WG11.3 Workshop on Database Security IV: Status and Prospects*, Elsevier Press, September 1990, pp.179-189.

[11] M.J. Nash, K.R. Poland, "Some Conundrums Concerning Separation of Duty", *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, IEEE Press, May 1990, pp.201-207

[12] K. Konstantin, W. Harald, "Analyzing Separation of Duties in Petri Net Workflows", *Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security,* Springer Press, May 2001, pp.102-114.

[13] S. Perelson, R. Botha, J. Eloff, "Separation of Duty Administration", *South African Computer Journal*, 27:64-69, 2001.