Task-activity based access control for process collaboration environments

Yahui Lu, Li Zhang, Jiaguang Sun

College of Software, Shenzhen University, Shenzhen 518060, PR China
Key Laboratory for Information System Security of Ministry of Education China, School of Software, Tsinghua University, Beijing 100084, PR China

1. Introduction

There is an increasing acceptance of Service-Oriented Architecture (SOA) as a paradigm for integrating applications within and across organizational boundaries. SOA includes a family of XML-based protocols to describe the definitions and interactions of Web services. Web Services provide the interoperability of various applications running on heterogeneous platforms over the Internet. A Web Service is the name of an object with methods that can be invoked through an Internet connection. WSDL [1] describes the interfaces and usages of a particular Web service. SOAP (Simple Object Access Protocol) [2] is the primary transport mechanism to convey the requests and responses of Web services. WS-BPEL [3] and WS-CDL [4] provide the local and global view to describe the interactions and compositions of Web services.

Workflow management systems (WFMSs) have become promising solutions for organizations to automate their business processes [5]. As an executable process definition language, WS-BPEL defines the business process composed by Web services and the interactions between these Web Services. In the Web services world, the elements of traditional WFMSs, including processes, task executors (users), resources (data) etc, are all represented by Web services. The relationships between these elements are all represented by the interactions of Web services and business processes. Several collaborative systems based on Web services and workflow technologies have been proposed [6–15].

Web services and business processes are important enterprise resources which need to be protected for authorized access only. Some standards for securing Web services have been proposed under SOA. WS-Security [16] describes the enhancement of the SOAP messaging to provide securities through message integrity, confidentiality, and authentication. Security Assertion Markup Language (SAML) [17] is an XML standard for exchanging authentication and authorization data between security domains. EXtensible Access Control Markup Language (XACML) [18] provides an access control framework for expressing access control policies and authorization decision procedures. Many access control models and mechanisms also have been proposed for WFMSs [19–24] and for collaborative systems [25–35].

However, despite these efforts, some specific access control requirements should be addressed under the new collaborative paradigm based on Web services and workflow technologies. In such process collaborative environments, each organization has its own WFMS. The collaborations among these organizations are fulfilled by interactions of processes between these WFMSs. To better clarify such process collaborative environments, in this paper, we first explore process collaboration from the view of access control and summarize several process collaboration patterns based on the consideration of permission assignment, withdrawal and transmission. To support these process collaboration patterns, four specific access control requirements are
proposed, including dynamic assignment of permissions, transmission and management of permissions between domains, separation of permission assignment and withdrawal, and duplex permission assignments.

To meet such requirements, we propose a task-activity based access control (TABAC) model. Although task and activity are often synonymous terms in many traditional WfMSs, in this paper, we differentiate these two terms in our TABAC model. Business process is composed by activities while dynamic permissions are related to tasks. Business process and authorization rules are linked by the relationships between tasks and activities. A task can be related with several activities with different operation commands for that task. By these commands, task permissions can be dynamically assigned to and withdrawn from processes during the interaction of activities. To transmit the task permissions and access control information between different processes, a SOAP based interaction protocol is also proposed. Finally, we describe the architecture of the collaborative system which implements TABAC in workflow management system and conforms to WS-BPEL and XACML specifications. TABAC model can flexibly meet the access control requirements for process collaboration environments.

This paper is organized as follows. Section 2 describes the related work. Section 3 describes a scenario for process collaboration and concludes some process collaboration patterns. Section 4 analyzes the access control requirements in process collaboration environments. Section 5 presents the task-activity based access control model. Section 6 describes the implementation of TABAC model. Section 7 concludes our work and proposes further work.

2. Related work

The emerging of Web services and workflow technologies has enabled a new paradigm for collaborative systems among different organizations. Hoffner et al. [6] proposed the CrossFlow architecture, in which the interactions between service consumers and providers are specified by contracts and implemented by business processes. Fan et al. [7] proposed an architecture for cross-enterprise business integration. The architecture is composed of three layers, including agreement management layer, service management layer and workflow management layer. Yan et al. [8] presented an inter-organizational workflow model (CA-PLAN) and presented a cooperative Agentflow WfMS system by Java machine and remote call process mechanism. Chiu et al. [9] proposed the concept of workflow view as a fundamental support mechanism for the interoperability of multiple workflows across business organizations. Chebbi et al. [10] present CoopFlow, a workflow cooperation framework, supporting dynamic plugging and cooperation between heterogeneous WfMSs. Shen and co-workers [11] presented an agent-based workflow model to support inter-enterprise collaboration by both internal processes and ad hoc external processes. Lee et al. [12] proposed a process-centric framework to service orchestration and choreography, and conversations over processes. Gong et al. [13] proposed a conceptual model of business process collaboration and proposed a semantic agent-based framework to facilitate business process collaborations. The cross-organizational collaborative commerce based on workflow system is discussed in [14]. The workflow system for cooperative grid computing is discussed in [15]. These literatures proposed several different models and architectures for the inter-organizational collaborations by business processes and Web services. However, little attention has been paid on the security issues in these models and architectures.

Access control is an important mechanism to protect workflow data for WfMSs. Atluri’s WAM [19] is the first workflow access control model. In this model, an Authorization Template (AT) is associated with each task. When the task actually starts execution, AT is used to derive the actual authorization rules for the task. Thomas and Sandhu [20] developed the Task Based Access Control model. They use an authorization step (AS) to indicate the execution of a task. Permissions are defined on each authorization step and user can get permissions from an authorization step. Wu et al. [21] proposed a comprehensive data authorization and access control mechanism for WfMSs, in which permissions are assigned to tasks and roles are introduced as a medium between users and tasks. Oh et al. [22] proposed the task-role-based access control (TRBAC) model. TRBAC model integrates workflow and non-workflow access control to give a complete access control model in an enterprise environment. Mattas et al. [23] discussed the dynamic and decentralized administration of access control in workflow systems. Wainer et al. [24] extended RBAC with delegation in the workflow context. In their model delegations can be specified and revoked between users. These workflow access control models mainly focus on access control within a single WfMS. They did not consider the access control requirements for inter-organizational collaborations by multiple WfMSs.

The first access control model for collaborative system was proposed by Shen and Dewan [25]. They developed a general framework based on access matrix model for supporting access control in multi-user interfaces. After that, various access control models are proposed for collaborative systems. Tolone et al. [26] give a comprehensive overview of access control models and mechanisms for collaborative systems. Kapsalis et al. [27] presented a context-aware access control which is based on RBAC model and incorporates dynamic context information in the form of context constraints. Lu et al. [28] discussed the domain based administration of RBAC in collaborative environments. Chen et al. [29] proposed a Virtual Enterprise Access Control (VEAC) model to handle resource management and sharing across each participating enterprise. It consists of two parts: a Project-based Access Control (PBAC) sub-model to manage public resources and a RBAC sub-model to manage private resources. Alsuaiman et al. [30] proposed Threshold-based Collaborative Access Control (T-CAC). In this model, every permission is associated to thresholds. A user with a role can contribute a specified weight to gain the permission. Vela et al. [31] proposed the Web service role-based access control architecture and discussed its implementation in a service-oriented architecture.

As for the inter-organizational collaboration by workflow and business processes, several access control models also have been proposed. Coulouris et al. [32] proposed the access control model based on roles and tasks to support the cooperation between principals in a virtual enterprise (VE). Ahn et al. [33] proposed a mechanism to enforce RBAC models for web-based inter-organizational workflow systems. Roles are used as an interface between workflows and security infrastructures specific to organizations. Kang et al. [34] present an access control solution for inter-organizational workflow. The solution includes two parts: an organization-specific access control module and a task-specific access control module. Li et al. [35] described different interaction interfaces in collaborative commerce to resolve multilevel access control policy conflicts. The researches in [32–35] can only support the situation in which permissions are assigned and withdrawn in one workflow activity. They cannot support the asynchronous interaction situations in which permissions are assigned and withdrawn in different workflow activities. The detailed explanations are described in Section 4.

3. Process collaboration

In this section, we first describe a scenario of enterprise collaboration by WS-BPEL processes. Then we summarize some
process collaboration patterns based on the consideration of permission assignment, withdrawal and transmission.

3.1. Introduction to WS-BPEL

WS-BPEL defines the model and grammar for describing the behavior of a business process based on interactions between the process and its partners [3]. The interaction with each partner occurs through Web Service interfaces. The WS-BPEL process defines how multiple service interactions with their partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination.

The actual business logic is represented as a group of activities, which are executed in a structured way. Activities are divided into two categories: basic activities and structured activities. Basic activities describe the elemental steps of the process behavior. Basic activities include <invoke>, <receive>, <reply>, <assign>, <wait>, <empty> etc. Structured activities encode control-flow logic, and therefore can contain other basic and structured activities recursively. Structured activities include <sequence>, <if>, <while> and <flow> etc.

The basic activities for interactions include <invoke>, <receive> and <reply>. The <invoke> activity allows the business process to call Web Services offered by service providers. The operations of <invoke> activity can be request-response or one-way operations, corresponding to WSDL 1.1 operation definitions. One-way invocation requires only the input variable since a response is not expected as part of the operation. Request-response invocation requires both an input variable and an output variable. In the request-response case, the <invoke> activity completes when the response is received. The <receive> activity allows the business process to wait for a matching message to arrive. The <receive> activity completes when the message arrives. The <reply> activity allows the business process to send a message in reply to a message that was received by an inbound message activity. The combination of a <receive> activity and a <reply> activity forms a request-response operation on a WSDL portType for the process.

3.2. A scenario

Fig. 1 illustrates a scenario of collaborative manufacture by three enterprises. Enterprise A sells some manufacturing products. Its namespace is “xiamen.kinglong.com”. Enterprise B is a manufacturing factory. Its namespace is “suzhou.kinglong.com”. Enterprise C sells the standard parts. Its namespace is “beijing.tsingnet.com”. Enterprise A includes two Web services: product service Sa (with ReceiveOrder and ManuCallBack operations) and design file access service Sd (with read and write operations). Sa is implemented by a product process Pa. Enterprise B includes two Web services: manufacture service Sb (with ReceiveOrder and ProgressQuery operations) and manufacture file access service Sm (with read and write operations). Sb is implemented by a manufacture process.
P. Enterprise C includes one Web service: part sales service Sc (with ReceiveOrder operation). Sc is implemented by a sales process Pc.

These enterprises interact with each other in the following steps:

1. When Enterprise A receives an order from Sa service, it first designs the product according to the customer’s requirements and invokes the Sc service to save the design file.
2. The product includes two parts: the standard part p artS and the specific part partM. Enterprise A buys p artS from Sc service of Enterprise C. It will also place an order to Sb service of Enterprise B to manufacture partM. During the manufacture, Enterprise A allows Sb to read the design file. Enterprise A will also query about the manufacture progress of partM to make sure that the production time will not be delayed.
3. When Enterprise B receives the order from Sb service, it first designs the manufacture file of partM according to the design file in Enterprise A. Then it begins to manufacture partM. During the manufacture, it may allow Sa to read the manufacture file of partM and to call the ProgressQuery operation to get the manufacture progress.
4. When Enterprise C receives the order from Sc service, it prepares the partartS from its storehouse and ships it to Enterprise A.
5. When Enterprise A receives the parts from B and C, it assembles them together and finally completes the customer’s order.

In this scenario, each participant provides several services as resources for other participants. Some services, such as Sa, Sb and Sc, are public services that can be accessed by any other participants. These services can receive orders from any other participants. Other services, such as Sd and Sm, are protected services which only can be accessed by authorized users. The Sd service should be only accessed by users who are executing the design tasks or manufacture tasks in process Pa. The users who are executing the design task of process Pa can read or write the design file by invoking the Sd service. The users who are executing the manufacture task of process Pa can read the design file by invoking the Sd service. No other users are allowed to invoke the Sd service and to access the design file. Also notice that the privileges to access the Sd service can only be allowed during the design and manufacture activities. Similarly, the Sm service should be only accessed by users who are executing the design tasks in process Pb, or by users who invoke the Sb service. An access control mechanism is required to guarantee that these protected services can only be accessed by authorized users as the specified manners.

3.3. Process collaboration patterns

We have shown that access control is an important aspect in process collaborations by the scenario of Section 3.2. From this process collaboration scenario, we can find that the collaborative mode between Enterprises A and B have some differences with the collaborative mode between Enterprises A and C. To further explore the relationships between access control and process collaboration, in this section, we describe some patterns about process collaborations from the view of access control. In previous researches, [37] presents workflow patterns within a business process. [38] presents service interaction patterns among Web services and business processes. [39] describes the patterns for inter-organizational business process collaborations from the new concept of sourcing. Different with these previous work, our research mainly focus on the exploration of process collaboration patterns from the view of access control.

Fig. 2 illustrates the process collaboration patterns that we have summarized based on the consideration of permission assignment, withdrawal and transmission. In these patterns, two processes P and Q interact with each other. P and Q are located in two different domains and supported by two different workflow engines. Permission p and p’ are belong to these two domains separately and will be assigned to the processes during the interactions. These patterns are classified into four categories.

1. Collaboration without permission assignment.
   • (a) Synchronous interaction: Process P invokes process Q to fulfill a job by an invoke request-response activity and then waits for the result. After Q completes its job, it will submit the result by a reply activity. The interaction between Enterprises A and C in the scenario of Section 3.2 is belong to this pattern.
   • (b) Asynchronous interaction: Process P invokes process Q to fulfill a job by an invoke one-way activity. Then P continues to execute other activities. After Q completes its job, it will invoke the callback operation of P to submit the result.
   • (c) Notification: Process P calls process Q by an invoke one-way activity. Q does not need to reply to P.

2. Collaboration with forward permission assignment. When process P invokes Q, it will assign permission p to Q. Then Q can access the resources allowed by p within P’s domain. There are three patterns according to the different manners of permission withdrawal.
   • (d) Withdraw implicitly: Process P assigns permission p to Q by an invoke request-response activity. When Q submits the result, permission p is withdrawn or expired automatically. Then Q can not access the resources related with p any longer.
   • (e) Withdraw synchronously: After process Q submits its job by an invoke request-response activity, it will wait for a response message from P. Process P will send the withdraw command to Q with the response message by a reply activity.
   • (f) Withdraw asynchronously: Process Q submits its job by an invoke one-way activity. Process P will send the withdraw command to Q with a request message by another invoke one-way activity.

3. Collaboration with backward permission assignment. Process Q will assign permission p’ to process P to allow P to access the resources related with p’ within Q’s domain.
   • (g) Backward permission assignment synchronously: Process P invokes process Q by an invoke request-response activity. Q replies a response message which assigns permission p’ to P. When Q submits the job to P, it also withdraws permission p’ from P.
   • (h) Backward permission assignment asynchronously: Process P invokes process Q by an invoke one-way activity. Q assigns permission p’ to P by another invoke one-way activity. When Q submits the job to P, it also withdraws permission p’ from P.

4. Collaboration with duplex permission assignment. Two processes assign permissions to each other. Process P assigns permission p to Q and Q assigns permission p’ to P.
   • (i) duplex permission assignment synchronously: Process P and Q interact with each other by invoke request-response activities. Permission p is assigned to Q in the request message of invocation and withdrawn in the reply message of submission. Permission p’ is assigned to P in the reply message of invocation and withdrawn in the request message of submission. The interaction between process P and P in the scenario of Section 3.2 is belong to this pattern.
   • (j) duplex permission assignment asynchronously: Process P and Q interact with each other by invoke one-way activities. The permissions are assigned and withdrawn in the request messages.
Here are some notes on these process collaboration patterns. First, pattern (d) is widely used in the traditional workflow access control models. In these models, permissions are related with a task (or an activity) of Process P and will be assigned to user (or process) Q when the task (or activity) is executed by Q. Permissions are withdrawn automatically after the task (or activity) is submitted by Q.

Second, we only summarize some basic process collaboration patterns. Many complex process collaborations, such as collaborations with multi-process interactions or multi-permission assignments, can be composed by these basic collaboration patterns.

Third, $p$ and $p'$ are just abstract representations for permissions. In practical applications, $p$ and $p'$ can be replaced by capabilities,
4. Access control requirements

Access control in collaborative system has been studied for decades. Some general requirements for access control in collaborative systems have been summarized as follow [26]:

- **Transparent:** Access control model must facilitate transparent access for authorized users and exclusion of unauthorized users in a flexible manner that does not constrain collaboration. Access control-produced effects on the rest of the system must be clear and easy to understand.
- **Granularity:** Access control model must be able to protect information and resources of any type and at varying levels of granularity.
- **Expressiveness:** Access control model must be expressive enough to specify complex access policies at different levels of detail.
- **Scalability:** Access control model must be able to support much richer quantity of operations than tradition single user models.
- **Runtime change of policy:** Access control model must support to specify and change policies at runtime depending on the environment.
- **Context:** Access control model may grant access by considering the current context of subject and object.

However, under the new collaborative paradigm based on Web services and workflow technologies, some additional access control requirements must be addressed. We have summarized several process collaboration patterns to reveal the relationship between access control and business process in Section 3.3. To support these process collaboration patterns, we can summarize the following access control requirements for process collaboration environments.

- **ACR1:** dynamic assignment of permissions. During process interactions, the permissions of each process may be assigned dynamically. The access control model should support this dynamic assignment of permissions directly.
- **ACR2:** transmission and management of permissions between domains. In traditional centralized WfMSs, the user and process are located in one domain and supported by one workflow engine. The user will be directly assigned with the permissions by workflow system. There is no need to transmit the permissions between the user and process. While in process collaboration environments, two processes may locate in two separate domains and supported by two different workflow engines. If a process attempts to access the resource in another domain, it should carry a certificate issued by that domain. Therefore, a mechanism should be defined to transmit and manage the certificates of permission between separate domains.
- **ACR3:** separation of permission assignment and withdrawal. In traditional WfMSs, permissions are related with one workflow activity (or task). When a user is executing an activity (or task), it also gets the permissions related with that activity (or task). Pattern (d) represents this situation. While in process collaboration environments, permission may be assigned by one activity and withdrawn by another activity. Permission can be hold within the period between these two activities. Therefore, the relationship between permission and process activity must be clearly defined in the access control model.
- **ACR4:** duplex permission assignments. In traditional WfMSs, only processes can assign permissions to users while users can not assign permissions to processes. The process and the user have different positions in the interactions. While in process collaboration environments, all the participated processes have the same position in the collaboration. Each process may assign permission to its partners. The permissions of each process may change dynamically during the interactions. Therefore, the access control model should support for duplex permission assignments.

By these requirements, the Role-based access control (RBAC) models [36] and their variants [27–31] are not fit for process collaboration environments. In RBAC, roles are assigned to user statically. Once a user has been related with a role, it can get the permissions with that role at any time. Although the concept of session is introduced into RBAC models to enable the user to possess different roles in different sessions, it still can not satisfy the requirements we have proposed. First, in RBAC, the session roles are just a subset of roles assigned to a user. The user can only select the roles within its assigned roles. While in process collaboration environments, any permission can be given to the user by its interactive partners. Second, in each session, the user can freely choose the session roles as its own will. While in process collaboration environments, the dynamic permissions of a process are given by its partners. The process itself can not choose its dynamic permissions. Therefore, as our viewpoint, the RBAC models are best fit for static permission assignment relationships. The introduction of sessions, contracts and contexts in RBAC just adds some dynamic characters within a static scope.

In fact, most access control models in WfMSs are task (or activity) based access control (TBAC). In TBAC, permissions are assigned to tasks and users can only get the permissions within the execution of tasks. Once the user completes and submits the task, it has no longer possessed the permissions to access the resources related to the task. TBAC is a dynamic access control mechanism: the user’s permissions can change dynamically by its status during the running of workflow process. The researches in [19–23] and [32–34] are all task based access control. However, these models mainly focus on support for pattern (d). They can not be directly used to support patterns (e)–(i). Therefore, a new access control model is needed for collaborative system based on Web services and multiple WfMSs.

5. Task-activity based access control

**Definition 1.** The Task-activity based access control model is composed by the following components:

- **WS:** the set of Web services.
- **OP:** the set of operations of Web Services.
- **P:** the set of permissions. \( P \subseteq 2^\text{WS}\setminus\text{OP} \). The permissions are defined as the operations on the Web services.
- **U:** the set of uses. The user can be a Web service or a human being. \( \text{WS} \subseteq U \).
- **R:** the set of roles.
- **T:** the set of tasks.
- **BP:** the set of processes.
- **ACT:** the set of activities in the process. These activities include invoke, receive, reply and other activities defined in WS-BPEL.
- **COM:** the set of operation commands for the task. The command can be assign, withdraw or assign–withdraw.
- **BPWSA \subseteq \text{WS} \times \text{BP} :** a relation between business process and its related Web service.
- **URA \subseteq U \times R:** a many-to-many user to role assignment relation. \( (u, r) \in URA \) denote a user can have the permissions with a role in RPA.
- **TAR \subseteq \text{BP} \times \text{ACT} \times T \times \text{COM} :** the task-activity relationship.
Fig. 3 illustrates the basic elements and their relationships of TABAC. In TABAC, permissions are defined as the operations on the Web services. As we have analyzed in Section 4, roles are fit for static permission assignments and tasks are fit for dynamic permission assignments. Therefore, we define that permissions can be assigned to tasks, roles and users to support the different access control requirements. User can get permissions by UPA directly, or by its roles (though URA and RPA), or by the tasks that it possesses dynamically during the interactions (through TPA).

Business process is defined by activities. Business processes can collaborate with each other by interactions of activities. Business processes and authorization rules are linked together by two relationships: BPWSA and TAR. BPWSA defines the relationships between Web services and business processes. Web services are implemented by business processes. TAR defines the relationships between tasks and activities. A task can be related with several activities with different operation commands for that task. Although task and activity are often synonymous terms in many traditional WIMSSs, in this paper, we differentiate these two terms in our TABAC model.

The command in TAR can be assign, withdraw or assign-withdraw. \((p, a, t, \text{assign}) \in \text{TAR}\) denotes that activity \(a\) in process \(p\) will assign task \(t\) to its interactive partner. \((p, a, t, \text{withdraw}) \in \text{TAR}\) denotes that activity \(a\) in process \(p\) will withdraw task \(t\) from its interactive partner. For \((p, a, t, \text{assign–withdraw}) \in \text{TAR}\), it is used for process collaboration pattern (d). The activity \(a\) must be an invoke request–response activity in this case. Activity \(a\) in process \(p\) assigns task \(t\) to its partner once its partner receives the request message. When the response message is received by activity \(a\), task \(t\) is withdrawn automatically and process \(p\) does not need to send an additional withdraw command to its partner. By these different commands defined between activities and tasks in TAR, all the process collaboration patterns can be supported by TABAC model.

**Definition 2.** An access control policy is a tuple \((\text{UPA, RPA, TPA, BPWSA, URA, TAR, C})\). Access control policy can be divided into two parts:

- Access check policy: Access check policy is a tuple \((\text{UPA, RPA, TPA, C})\). It defines the rules to decide the user's permissions.
- Process security policy: Process security policy is a tuple \((\text{BPWSA, URA, TAR, C})\). It defines the access control information related with business processes.

An access control policy can be composed by the access check policy and the process security policy. We give examples on the access check policy and the process security policy.

**Example 1.** The access check policy of Enterprise A is defined as the following. We use \(*\) to denote that the rule can be matched by any entities. By this policy, any user can invoke the ReceiveOrder operation of Sa service. Role RB can not invoke write operation of Sd service. The user with TADesign task can invoke Sd service to read or write the design file. The user with TManu task can invoke the read operation of Sd service and the ManuCallback operation of Sa service.

\[
\text{SacA:} := \text{(UPA{(*, ReceiveOrder, Sa, Permit)}, RPA{(RB, write, Sd, Deny)}, TPA{(TADesign, read, Sd, Permit), (TADesign, write, Sd, Permit), (TManu, read, Sd, Permit), (TManu, ManuCallback, Sa, Permit))}}
\]

**Example 2.** The process security policy of Enterprise A is defined as the following. By this policy, the service Sa is implemented by business process Pa. Sa is assigned to role RA. In process Pa, the task TADesign can be assign to users when the Design activity is executed and should be withdrawn after the Design activity is completed. The task TManu can be assigned to users in the ManufacturePartM activity and should be withdrawn in the Acknowledge activity.

\[
\text{PSPA::=} \text{(BPWSA{(Sa, Pa)}}, \text{URA{(Sa, RA)}}, \text{TAR{(Pa, Design, TADesign, Assign–Withdraw), (Pa, ManufacturePartM, TManu, Assign)}, (Pa, Acknowledge, TManu, Withdraw))}}
\]
6. Design and implementation

In this section, we describe the implementation and application of TABAC model in collaborative systems. We first propose a SOAP based interaction protocol to transmit the task permissions between processes. Then we will describe the architecture of the system which implements TABAC in workflow management system. We will also describe the application of the system to the scenario of Section 3.2.

6.1. Interaction protocol

Since interactive processes may locate in different organization domains, it is necessary to define an interaction mechanism to transmit the task permissions and their management commands between these domains. The attributes of each process also need to be transmitted for access check. SOAP provides the primary transport mechanism to convey the requests and responses message of the interactive parties. Therefore, SOAP message can be used for transmission of security information of TABAC model. We extend SOAP header for this purpose.

As Fig. 4 shows, there are two parts in the <Security> tag of SOAP header. The first part is <SubjectInfo> tag. It describes a subject’s security attributes, including its userID, its roles and its task tokens. This part is used to check the permissions of the subject by the authorization decision procedure. The second part is the <taskTokenManage> tag. It includes the <taskToken> and <command> tag. The task in TABAC model is represented by the <taskToken> tag. A <taskToken> tag is composed by a <taskName> tag to describe the name of the task, a <processInstanceId> tag to describe which process instance issues this task token, and a <timestamp> tag to guarantee the time validity of the task token. The <command> tag is used to describe the command on the task token. The command can be “assign”, “withdraw” or “assign–withdraw” as defined in TABAC model.

Example 3. Fig. 4(a) shows the SOAP message which process Pa sends to Enterprise B to invoke the ReceiveOrder operation of service Sb. It assigns a task token TManuToken1 to Sb. Fig. 4(b) shows the SOAP message which Pb sends to Enterprise A to invoke Sd with the task token TManuToken1.

6.2. System architecture

We extend ActiveBPEL with the implementation of TABAC model to support inter-organizational collaborations. ActiveBPEL [40] is an open source workflow engine that is capable of executing process definitions created for the WS-BPEL. We also use XACML specification to provide the access control framework for expressing the TABAC access check policies. Fig. 5 illustrates the architecture of the system. It includes three layers: transportation layer, access control layer and workflow layer.

6.2.1. Transportation layer

The transportation layer is responsible for transmission of SOAP messages. We use HTTP as the underlying transportation mechanism for SOAP messages. This layer includes two modules: Http Server and Http Client. Http Server provides the access to Web service by URL. The message processing procedure of this layer is as following:

1. When Http Server module receives a HTTP request message, it will extract the SOAP request message and sends it to access control layer. If this message is an authorized access, it will be sent to workflow layer and received by a receive activity.

   <SoapEnvelope
   xmlns:Soap="http://www.w3.org/2001/12/soap-envelope"
   xmlns:ws="http://security.tabac.org">
   <Soap:Header>
     <ws:SubjectInfo>
       <ws:userID>Pa</ws:userID>
       <ws:roleSet>Manager</ws:roleSet>
     </ws:SubjectInfo>
   </Soap:Header>
   <Soap:Body>
     <ns1:ReceiveOrder xmlns:ns1="http://suzhou.kinglong.com/Sb">
       <ns1:processName>ReceiveOrder</ns1:processName>
       <ns1:taskToken>TManuToken1</ns1:taskToken>
     </ns1:ReceiveOrder>
   </Soap:Body>
 </SoapEnvelope>

2. When Http Server module receives a SOAP response message from workflow layer or access control layer, it will send out this message by HTTP response. This message may be a response message from the reply activity in workflow layer or a fault message from access control layer.

3. When Http Client module receives a SOAP request message from workflow layer, it will send out this message by a HTTP request. This message is from an invoke activity in workflow layer.

4. When Http Client module receives a HTTP response message, it will extract the SOAP response message and sends it to workflow layer. This message is a reply message to the invoke request-response activity in workflow layer.
6.2.2. Access control layer

The access control layer is used to enforce the run-time access control mechanism. This layer includes Policy Enforcement Point (PEP), Policy Decision Point (PDP), Combination Algorithm and Attribute Finder modules. PEP constitutes the point where the policy decisions are actually enforced. PDP is responsible for decisions regarding access permission to the requested services. Combination Algorithm module is responsible for giving a final decision when several policy rules are matched for a single request. Attribute Finder module is to get the status of each task token from the Attribute Provider module in the workflow layer.

In TABAC model, an access request may match the access control rules by its userID, its roles or its tasks. We defines that task has the first precedence to match the access control rules and role has the last precedence. If several rules are matched with an attribute, the Combination Algorithm is called to decide the result of these rules. The message processing procedure of this layer is as following:

1. As soon as PEP receives a SOAP request message for a service access from HTTP layer, it formulates an XACML request message from this SOAP message and sends the XACML request message to PDP.
2. When PDP receives an XACML request message from PEP, it decides the result by the following procedure and then return the result to PEP.
   a. For each task token in \(<\text{taskTokenSet}>\) tag of the request message
      i. Call Attribute Finder module to get the status of the task token from workflow layer
      ii. If the task token is expired or not exist, then return Deny
   b. Match with the rules in access check policy.
      i. If there are rules matched with task attribute, call Combination Algorithm module to decide the result of the rules. The result may be Permit or Deny. Return the result.
      ii. If there are rules matched with userID attribute, call Combination Algorithm module to decide the result of the rules. Return the result.
      (iii) If there are rules matched with role attribute, call Combination Algorithm module to decide the result of the rules. Return the result.
   c. If there is no matched rules, return NotApplicable.
3. When PEP receives the result message from PDP:
   a. If the result is Permit, PEP will transfer the SOAP request message to the workflow layer.
   b. If the result is Deny or NotApplicable, PEP will send a fault message back to the Http layer and reject the access.

6.2.3. Workflow layer

The workflow layer is to manage the business processes and their related permissions. It includes BPEL processor, Web Service Handler, SOAP Packager, SOAP Parser, Process Attribute Provider, Message Dispatch, Activity Manager, Task Manager and Process Manager. The Message Dispatch module is to schedule the incoming messages and outcoming messages. The Process Manager module is to manage the creation, execution, and destroy of processes and process instances by their BPEL definitions and WSDL definitions. The Activity Manager module is to manage the creation, execution and destroy of activities and activity instances. The Task Manager module is to handle SOAP header and manage the task tokens of each process instance. The Attribute Provider module is to provide the status of task tokens for other layers. The message processing procedure of this layer is as following:

1. When the workflow engine receives the SOAP message from SOAP parser, it first dispatches the incoming message to the correct process instance. If there is no related process instance and the request matches a start activity, a new process instance is created. After that, the Task Manager will be called to handle the \(<\text{taskTokenManage}>\) tag of this SOAP message.
2. When the workflow engine sends the SOAP message to the partner, it will call Task Manager to write the related information into the \(<\text{SubjectInfo}>\) and \(<\text{taskTokenManage}>\) tag of SOAP header.
Task Manager module is to handle the security information of SOAP header. Algorithm 1 describes this procedure. Two tables must be maintained to manage the task tokens for each process instance. The sendTaskToken Table is for the management of task tokens sent from each process instance. It is defined by tuple (taskToken, process, processInstance, activity, activityInstance, command). The receivedTaskToken Table is for the management of task tokens received by each process instance. It is defined by tuple (taskToken, process, processInstance, activity, activityInstance, command).

Algorithm 1. The algorithm of task management in Task Manager module

Input: message m, sendTaskToken Table st, receivedTaskToken Table rt, Process security policy psp

Output: message m

(1) If m is a received message
(2) Get the destination process p, process instance pi, activity a and activity instance ai of m;
(3) /*** implicitly withdraw task token in st ***/
(4) If Type(ai) == "invoke request-response" and psp.TAR.command(a) == "assign–withdraw" then
(5) delete records from st where st.activityInstance=ai;
(6) else
(7) /*** change the receivedTaskToken table rt ***/
(8) If m.taskTokenManage.command == "assign" then
(9) insert (m.taskToken, p, pi, a, ai, "assign") into rt;
(10) elseIf m.taskTokenManage.command == "withdraw" then
(11) delete record from rt where rt.taskToken=m.taskToken;
(12) elseIf m.taskTokenManage.command == "assign–withdraw" then
(13) insert (m.taskToken, p, pi, a, ai, "assign–withdraw") to rt;
(14) If m is a message to be sent
(15) Get the source process p, process instance pi, activity a and activity instance ai of m;
(16) /*** Fill the <SubjectInfo > tag of SOAP Header ***/
(17) m.SubjectInfo.userID = service name of process p in psp;
(18) m.SubjectInfo.roleset = roles of process p in psp;
(19) m.SubjectInfo.tasks = taskTokens from rt where rt.ProcessInstance = pi;
(20) /*** implicitly withdraw task token in rt ***/
(21) If Type(ai) == "reply" then
(22) Get the receive activity instance rai related with ai;
(23) If there exists records in rt where rt.activityInstance = rai and rt.command = "assign–withdraw" then
(24) delete these records;
(25) /*** Fill the <taskTokenManage > tag of SOAP Header ***/
(26) Get command com and task name t of activity a in process p from psp.TAR;
(27) If com == "assign" or com == "assign–withdraw" then
(28) creates a new task token tt=(t, pi, timestamp);
(29) insert (tt, p, pi, a, ai, com) into rt;
(30) m.taskToken = tt;
(31) m.command = com;
(32) elseIf com == "withdraw" then
(33) select record rec from st where st.activityInstance=ai;
(34) m.taskToken = rec.taskToken;
(35) m.command = com;
(36) delete record from st where st.activityInstance=ai;

6.3. Application and discussion

In this section, we describe how to implement the TABAC model and the related mechanisms to support the scenario in Section 3.2. Fig. 6 shows the architecture of collaborative systems. Each enterprise deploys a system as Fig. 5 shows, which includes four servers: HTTP server, access control server, workflow engine server and database server. The workflow engine server can support the process collaborations within and across these enterprises. The database server stores the sent and received task tokens of each process. Access control server provides the run-time access control mechanism for each enterprise. Each enterprise must define its own access control policy according to the TABAC model.

The access check policies for these enterprises are defined as the following.

SacA::=(UPA{(*, ReceiveOrder, Sa, Permit)}, RPA{(RB, write, Sd, Deny)}, TPA{(TADesign, read, Sd, Permit), (TADesign, write, Sd, Deny)}).
Permit), (TManu, read, Sd, Permit), (TManu, ManuCallBack, Sa, Permit)))
SacB::=(UPA([\{, ReceiveOrder, Sb, Permit]), RPA([RA, write, Sm, Deny]), TPA(TDesign, read, Sm, Permit), (TDesign, write, Sm, Permit), (TCustomer, read, Sm, Permit), (TCustomer, ProgressQuery, Sb, Permit)))
SacC::=(UPA([\{, ReceiveOrder,Sc,Permit])))

These policies provide the rules for access control modules to decide whether an access is permitted or denied. By the policy SacB, any user can invoke the ReceiveOrder operation of Sb service of Enterprise B. Role RA can not invoke write operation of Sd service. The user with TDesign task can invoke Sm service to read or write the manufacture file. The user with TCustomer task can invoke the read operation of Sm service and the ProgressQuery operation of Sb service. By the policy SacC, any user can invoke the ReceiveOrder operation of Sc service of Enterprise C. The policy SacA has already been explained in Example 1.

The process security policy for each enterprise is defined as the following.

PSPA::=(BPWSA((Sa, Pa)), URA((Sa, RA)), TAR((Pa, Design, TDesign, Assign–Withdraw), (Pa, ManufacturePartM, TManu, Assign), (Pa, Acknowledge, TManu, Withdraw)))
PSPB::=(BPWSA((Sb, Pb)), URA((Sb, RB)), TAR((Pb, ConfirmOrder, TCustomer, Assign), (Pb, Design, TDesign, Assign–Withdraw), (Pb, ProductComplete, TCustomer, Withdraw)))
PSPC::=(BPWSA((Sc, Pn)), URA((Sc, RC)), TAR())

These policies are used to define the access control information related with business processes. By the policy PSPB, service Sb is implemented by business process Pb. Sb is assigned to role RB. In the process Pb, the task TDesign can be assign to users when the Design activity is executed and should be withdrawn after the Design activity is completed. The task TCustomer can be assigned to users in the ConfirmOrder activity and should be withdrawn in the ProductComplete activity. By the policy PSPC, service Sc is implemented by business process Pn. Sc is assigned to role RC. The policy PSPA has already been explained in Example 2.

Now we describe how these policies can work together with access control modules and workflow engines to support the task transmissions and managements among enterprises. It is worthy to note that each process can be instantiated to several process instances in run time. It is these process instances that interact and collaborate with each other. Suppose Palns1 is one of the instances of process P a and PbIns1 is one of the instances of process Pb.

(1) In Enterprise A, when the process instance Palns1 is executed to the activity ManufacturePartM, it will invoke the service Sb of Enterprise B by a soap message. According to the TAR rule (Pa, ManufacturePartM, TManu, Assign) in the policy PSPA, a task token of task TManu will be attached to the header of this soap message. According to Algorithm 1, the task manager module of Enterprise A will create this task token, fill this token in the header of the soap message, and records the token in the sendTaskToken table. Suppose the name of this task token is TManuToken1. The format of this soap message is shown in Fig. 4(a).

(2) When this soap message arrives at Enterprise B, it will be checked by access control module. According to the UPA rule ([\{, ReceiveOrder, Sb, Permit]) in the access check policy SacB, any user can invoke the ReceiveOrder operation of service Sb. Therefore, this invocation will be permitted and the soap message will be sent to the workflow engine. Since Sb is implemented by process Pb, a new process instance PbIns1 will be created. PbIns1 will receive the soap message and save the task token TManuToken1 in the receivedTaskToken table. This task token will assign task TManu to PbIns1.

(3) When PbIns1 is executed to the activity ConfirmOrder, it will send a soap message to reply the invocation of Palns1. According to the TAR rule (Pb, ConfirmOrder, TCustomer, Assign) in the policy PSPB, a task token of task TCustomer will be attached in the header of this soap message. The task manager module of Enterprise B will create this task token, fill this token in the header of the soap message, and records the token in the sendTaskToken table. Suppose the name of this task token is TCustomerToken1. This soap message and the task token will be received by Palns1. The task token will assign task TCustomer to Palns1.

(4) After this stage, when Palns1 invokes Sd to read the design file in the activity GetDesignFilePartM, it will attach the task token TManuToken1 in the header of the soap message. According to the TPA rule (TManu, read, Sd, Permit) in SacA, users with task TManu can be allowed to invoke the read operation of service Sd. Therefore, this invocation will be permitted and the soap message will be transferred to the service Sd. Similarly, when Palns1 invokes the ProgressQuery operation of Palns1 with task token TCustomerToken1, the access control module in Enterprise B will check the soap message. According to the TPA rule (TCustomer, ProgressQuery, Sb, Permit), the invocation is permitted and the soap message will be transferred to PbIns1.

(5) When PbIns1 is executed to the activity ProductComplete, a soap message will be sent from PbIns1 to Palns1. According to the TAR rule (Pb, ProductComplete, TCustomer, Withdraw) in the policy PSPB, a command which withdraws the task token TCustomerToken1 will be attached to the header of this soap message. The task manager module of Enterprise B will delete the task token TCustomerToken1 from its sendTaskToken table. After Palns1 receives the soap message, the token TCustomerToken1 will be deleted from its receivedTaskToken table by task manager module in Enterprise A.

(6) When Palns1 is executed to the activity Acknowledge, a soap message will be sent from Palns1 to PbIns1. According to the TAR rule (Pa, Acknowledge, TManu, Withdraw) in the policy PSPA, a command which withdraws the task token TManuToken1 will be attached to the header of this soap message. The task manager module of Enterprise A will delete the task token TManuToken1 from its sendTaskToken table. After PbIns1 receives the soap message, the token TManuToken1 will be deleted from its receivedTaskToken table by task manager module in Enterprise B.

(7) When this soap message arrives at Enterprise B, it will be checked by access control module. According to the UPA rule ([\{, ReceiveOrder, Sb, Permit]) in the access check policy SacB, any user can invoke the ReceiveOrder operation of service Sb. Therefore, this invocation will be permitted and the soap message will be sent to the workflow engine. Since Sb is implemented by process Pb, a new process instance PbIns1 will be created. PbIns1 will receive the soap message and save the task token TManuToken1 in the receivedTaskToken table. This task token will assign task TManu to PbIns1.

By these mechanisms and policies deployed in each enterprise, the specific access control requirements can be flexibly satisfied to support the process collaborations.

- **ACR1:** dynamic assignment of permissions. In TABAC, permissions can be related with task to support the dynamic permission assignments. Permissions can also be related with role to support the static permission assignment and related with user to support the fine grained permission assignment. User can get permissions by its userID, its roles and its tasks. The advantage of this method is that it is more flexible to support the various
access control requirements. The disadvantage is that it is more complex for authorization decision procedure and the policy management.

- **ACR2**: transmission and management of permissions between domains. In TABAC, dynamic permissions are related with tasks. We propose a SOAP based process interaction protocol to support the transmission of task tokens between domains. The operation command for each task token is also transmitted with the SOAP message. In the implementation, a task manager module is developed to manage these task tokens for each process instance.

- **ACR3**: separation of permission assignment and withdrawal. In TABAC, the relationship TAR is defined to specify the commands for the task in process activities. The command can be assign, withdraw or assign–withdraw. By these commands, task permissions can be assigned to the interactive partner by one activity and withdrawn by another activity. The partner can hold the task permissions within the period between these two activities.

- **ACR4**: duplex permission assignments. In TABAC, each process can receive task permissions by receive activities and send task permissions by invoke and reply activities. The exchange of task token can be bidirectional by SOAP request and response message. In the implementation, the task manager module manages the task tokens that have been sent and received by each process instance.

7. Conclusion and future work

It is rather a great challenge to balance the competing goals of collaboration and security in collaborative environments. In this paper, we focus on the access control aspect to provide security for organizations in process collaboration environments. The contributions of our study are as follows:

1. Requirement aspect: we summarized ten common process collaboration patterns from the view of access control. We also concluded four specific access control requirements to support these patterns in process collaboration environments.

2. Model aspect: we proposed a task-activity based access control model. In this model, we differentiate the term task and activity. Business process is composed by activities and dynamic permissions are related to tasks. Task permissions can be dynamically assigned to processes by the interaction of activities.

3. Implementation aspect: we presented an architecture to implement TABAC model in workflow management system. A SOAP based interaction protocol is also proposed to transmit the task permissions between processes.

Our work can be extended in the following aspects. First, the TABAC model can be extended to support other important access control components, such as separation of duty (SOD) constraints, contexts etc. Second, SAML and PKI (public key infrastructure) can be introduced into the system for authentication of users and management of task tokens. Third, a thorough and detailed exploration on process collaboration patterns can help us to gain further insight into interactions and collaborations based on business processes.

Acknowledgements

The work is partially supported by the National Natural Science Foundation of China (Grant No. 60673008, Grant No. 60673122), the National High-Tech Research and Development Program of China (Grant No.2007AA042135) and the National Basic Research Program of China (Grant No. 2004CB719405).

References


Yahui Lu is a lecturer in College of Software at Shenzhen University in People's Republic of China. He received his PhD degree from Department of Computer Science and Technology at Tsinghua University in 2008. His current research interests are in information system security, formal methods, product lifecycle management systems and business process management systems.

Li Zhang is an associate professor in School of Software at Tsinghua University in People’s Republic of China. He received his PhD degree in Mechanical Manufacturing at Beijing University of Aeronautics and Astronautics in 2000. His current research interests are in virtual enterprise, product lifecycle management, business process management, modeling and analysis of business process and access control by formal methods.

Jiaguang Sun is a professor in School of Software and Department of Computer Science and Technology at Tsinghua University in People’s Republic of China. He is a member of the Chinese Academy of Engineering and Dean of School of Software. He is also President of Executive Council of China Engineering Graphics Society and Vice President of National Natural Science Foundation of China. He received the BS degree from Department of Automation, Tsinghua University in 1970. His current research interests are in Computer Graphics, Computer Aided Design, Enterprise Information System, software formal verification and software engineering.