SOAC-Net: A model to manage service-based business process authorization

Haiyang Sun
Macquarie University

Jian Yang
Macquarie University

Weiliang Zhao
University of Wollongong, wzhao@uow.edu.au

Sanjay K. Nepal
CSIRO

SOAC-Net: A Model to Manage Service-Based Business Process Authorization

Haiyang Sun, Jian Yang
Department of Computing,
Macquarie University, Australia
{*haiyang.sun, jian.yang*}*@mq.edu.au*

Weiliang Zhao
Faculty of Engineering,
University of Wollongong, Australia
wzhao@uow.edu.au

Surya Nepal
CSIRO ICT Centre, Sydney, Australia
Surya.Nepal@csiro.au

***Abstract*—Business process (BP) can be supported by a large number of resources with evolving contents. In order to receive the support from these resources, the BP must satisfy the authorization policies of these resources. On the other hand, a BP also has its own authorization policies that users must satisfy in order to interact with the BP. Meanwhile, execution policies need to be applied to manage the sequence of tasks invocations in a BP. Therefore, without proper coordination among these policies, BP may not be able to perform correctly, e.g., imperative support from a specific resource could be missing or unauthorized user access can occur. An effective authorization management bringing all types of policies together becomes a must for a BP executing correctly without breaking any authorization and business rules. In this paper, we propose a process model, SOAC-Net that is incorporated with an authorization model, Process-Aware Service-Oriented Authorization Control (*PASOAC*). *PASOAC* is an extension of Role Based Access Control (RBAC), which takes both resource and user into account. A set of authorization constraints are designed in *PASOAC* to coordinate the user access and the resource support in a process environment.**
***Keywords*: Authorization, Authorization Policy, Business Process.**

I. Introduction

A business process (BP) in modern organization can be operated in a distributed environment involving multiple parties with dynamic availability and a large number of resources with evolving contents. These resources can be various types of applications that come from different organizations, e.g., web services. Therefore, these resources can belong to different security domains, and have different security and interest requirements. In order to acquire the support from these resources, a BP must be able to satisfy these resources' authorization policies. On the other hand, interacting with user is imperative for the execution of BP. But the user can access the specific process functions, only after it can satisfy the process's authorization policies. Execution policies are used to manage the sequence of tasks invocations within a BP, i.e., business logic. Therefore, without coordination on these policies, a BP may not be able to perform properly. For example, supposing a task in a BP can only be invoked once, if no proper policies coordination is put in place, multiple invocations by the same user can occur. Supposing task A and task B are a pair of sequenced tasks in a BP. Multiple users can invoke task A; while task B can only be accessed by one user. A constraint is set up in the BP as the user who can invoke task B must have already invoked task A. Now if task A is accessed by a user who is not granted permission to invoke task B, then no any other user is allowed to access task B. So BP is discontinued. Such dependency between user accesses also need to be identified between resource supports, and even between user access and resource support which makes policy coordination complex. How to manage the user accesses and the resource supports in a BP in a distributed environment, e.g., web service domain, then becomes an issue that is not tackled yet. Let us take a motivating example.

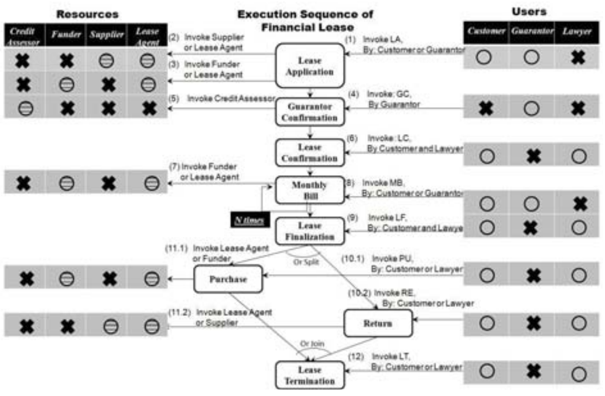
Resources
Credit Assessor
Funder
Supplier
Lease Agent
Execution Sequence of Financial Lease
Users
Customer
Guarantor
Lawyer
(1) Invoke LA, By: Customer or Guarantor
(2) Invoke Supplier or Lease Agent
(3) Invoke Funder or Lease Agent
Lease Application
(4) Invoke GC, By Guarantor
(5) Invoke Credit Assessor
Guarantor Confirmation
(6) Invoke LC, By Customer and Lawyer
Lease Confirmation
(7) Invoke Funder or Lease Agent
Monthly Bill
(8) Invoke MB, By: Customer or Guarantor
N times
Lease Finalization
(9) Invoke LF, By: Customer and Lawyer
(11.1) Invoke Lease Agent or Funder
Or Split
(10.1) Invoke PU, By: Customer or Lawyer
Purchase
(10.2) Invoke RE, By: Customer or Lawyer
(11.2) Invoke Lease Agent or Supplier
Return
Or Join
(12) Invoke LT, By: Customer or Lawyer
Lease Termination

Figure 1. Execution Sequence of Financial Lease

A. Motivating Example

Financial Lease is a BP that can provide the business equipment finance and leasing solutions in web service domain (See Fig. 1). When *Financial Lease* receives a lease application from a customer, it firstly assesses the value of the product that the customer wants to lease from a supplier and seeks a specific funder who can provide financial support. A guarantor for the customer must be confirmed to lower the risk of bad debt. Customer and his/her lawyer can confirm the lease application, once the credit history checks of customer and guarantor are finished. The repayment on the leased product will be paid regularly. After the agreed period, the customer or his/her lawyer can advice *Financial Lease* a lease finalization notice that enables *Financial Lease* to start the internal finalization process. The customer has two after-lease options, (1) purchase the product with a small-amount cost, or (2) return the product. Regarding to the purchase, funder will evaluate the cost to be paid by the customer; while for returning the product, supplier will carry out the maintenance check on the product. The whole lease process is terminated after the purchase or the return.

In Fig. 1, we illustrate an execution sequence of *Financial Lease*. Each operation (also known as task) of *Financial Lease* is depicted as a rectangle in the process. A *User* table is used to illustrate three types of users, `Customer, Guarantor,` and `Lawyer` that are depicted as the columns of the table, and each row of the table corresponds to specific operation in *Financial Lease*. A white circle in the table indicates that the type of user (Column) can access the operation (Row); while black cross means that the operation can not be accessed by specific user. Four types of resources, `Funder, Supplier, Lease Agent`, and `Credit Assessor`, are illustrated in *Resource* table in Fig. 1. The circles with horizon stripes in *resource* table represent that this type of resource (Column) can support the specific operation (Row); while black crosses in *Resource* table share the same semantics of black crosses in *User* table. The sequence of interactions among users, resources, and *Financial Lease* is numbered in Fig. 1 as a set of operation invocations. The requirements for accessing and supporting *Financial Lease* are described below,

- **User**: (1) `Customer` should be able to access all operations. (2)*Guarantor Confirmation* can only be made by `guarantor`. `Guarantor` can start the lease on behalf of the customer and help to repay the rental. (3) `Lawyer` after lease finalization stage can, on behalf of its client, deal with any rest activities. Furthermore, `lawyer` is necessary to involve in operations of *Lease Confirmation* and *Lease Finalization* with `customer`.
- **Resource**: (1) `Funder` is used to provide financial support. (2) `Supplier` is the product provider. (3) `Lease Agent` can provide both product and fund to *Financial Lease*, since the `lease agent` will seek its own funder and supplier. (4) `Credit Assessor` can evaluate the credit histories of `guarantor` and `customer`.

Based on the above example, we can observe that, (1) an operation within the BP can be accessed and supported by multiple types of users and resources, and (2) the inherently business logic (execution policy) set up within BP of *Financial Lease* requires the compliance of the user accesses and the resource supports. Hence, an effective authorization management should be performed on coordinating the user accesses and the resource supports in a process environment. Otherwise, authorization issues can be raised to cease process execution.

Restricting when the user accesses and the resource supports can/can't be used on specific operation during the execution of BP is an initial step to maintain the BP running properly. For example, supposing the `Guarantor` finishes accessing the operation *Guarantor Confirmation*, the permission to access this operation should be revoked immediately from the `Guarantor` to avoid repeated submissions of guarantor information. Furthermore, although multiple users or resources can access or support the same operations, not all of them can become effective which may suspend the execution of BP. In Fig. 1, *Monthly Bill* operation can be supported by `Funder` or `Lease Agent` and accessed by `Customer` or `Guarantor`. To avoid fraudulent activity, A `guarantor` can pay the rental on behalf of the `customer` to access the *Monthly Bill* operation only if the bill is issued by a `funder`, rather than a `lease agent`, i.e., during the execution of operation *Monthly Bill*, the invocation by `Guarantor` depends on the `Funder` support. This constraint is used when an entity can play as both `lease agent` and `guarantor`. Obviously, if the entity pays the bill issued by itself, it may eventually do harm to `customer`'s interest. `Funder` as the financial provider can not play as `guarantor`. Hence, the `Guarantor` can pay the bill issued from `Funder` only. In a summary, `Guarantor` becomes ineffective if the operation *monthly bill* is supported by `Lease Agent`. Supposing `Customer` also can not be used due to another dependency restriction, the BP will be suspended since both users, `Customer` and `Guarantor`, become unavailable to invoke operation *Monthly Bill*.

Therefore, we can conclude that, the user accesses and the resource supports are not only regulated by specific authorization policies to guide what the user can access and what the resource can support. They are also restricted by the business constraints enacted during the execution of business process to maintain the security in a process environment. These business constraints can be categorized as follows,

- **Synchronization**: The sequence of the user accesses and the sequence of resource supports should both synchronize with the execution sequence of the operations in BP. When an operation is ready to execute in a process instance, the relevant users and resources that can access and support the operation should be invoked. Once the operation finishes, the permissions assigned to user and resource to access and support the operation should be revoked immediately. The user and resource that can access and support the next operation in the process instance then should be ready. In above example, when the operation *Guarantor Confirmation* starts, the authorization to access the operation should be granted to `Guarantor`. When the operation finishes, the authorization of `Guarantor` access on the operation should be revoked immediately.

- **Dependency**: A user access or a resource support on specific operation in a BP may depend on another achieved user access or resource support. In above example, the access of the `Guarantor` on the operation of *Monthly Bill* depends on the support of the `Funder` on the same operation. If the `Funder` is used to support the *Monthly Bill*, then the `Guarantor` can be used to access the operation. Otherwise, the access of `Guarantor` on the operation of *Monthly Bill* should be restricted.

In a summary, to maintain the security during the execution of BP as motivating scenario, the accesses of three users and the supports of four resources should be synchronously sequenced with the BP of *Financial Lease*. Within the sequence of user accesses and the sequence of resource supports, each access and support must also satisfy the dependency requirements to cater for business security demands. Existing works, e.g., [7], [8], mainly focus on managing synchronization between the execution sequence of the operations and the sequence of user accesses. They do not take the sequence of resources supports into account. Furthermore, much representative research works, e.g., [9], design constraints for avoiding conflict of interest within the user accesses. Dependency constraint is still missing. Particularly, dependency constraints can not only be enabled between user accesses, but can also be executed between resource supports, and even between user access and resource support, which become more complex with the involvement of resource support management.

Hence, an authorization management based on the execution sequence of BP should be developed to maintain the security in a process environment. In this work, a conceptual model named Process-Aware Service-Oriented Authorization Control (PASOAC) is introduced to manage authorization of business process by considering both user access and resource support. Two types of authorization policies, (1) Authorization Synchronization Policies and (2) Authorization Dependence Policies, are included in the conceptual model to state the above business security demands. A process model SOAC-Net incorporated with PASOAC is designed to represent authorization flow. An authorization flow is the sequence of user accesses and the sequence of resources supports with associated authorization constraints, that is different from the control flow within normal workflow model, e.g., workflow-net [1]. The authorization flow can reflect the control flow of business process by enforcing synchronization policy. It can also enforce the dependency policy on top of the execution policy of the BP, which facilitates the authorization management on access control level only without delving into task execution sequence

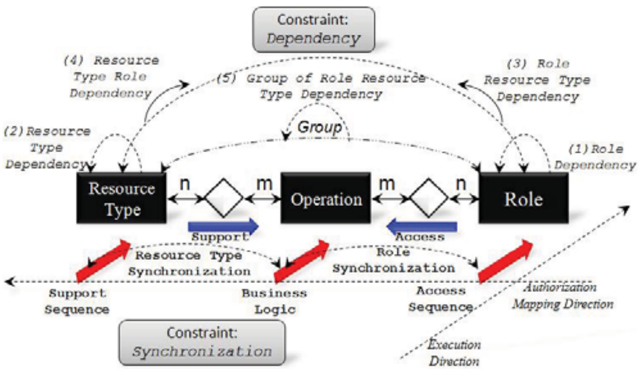
Constraint: Dependency
(4) Resource Type Role Dependency
(5) Group of Role Resource Type Dependency
(3) Role Resource Type Dependency
(2) Resource Type Dependency
Group
(1) Role Dependency
Resource Type
n
m
Operation
m
n
Role
Support
Access
Resource Type Synchronization
Role Synchronization
Support Sequence
Business Logic
Access Sequence
Authorization Mapping Direction
Constraint: Synchronization
Execution Direction

Figure 2. Process-Aware Service-Oriented Authorization Control

level.

The rest of paper is organized as follows. Section 2 describes the conceptual model PASOAC. Authorization policies are also defined in this section. The specification of SOAC-Net is described in Section 3. Section 4 overviews some related work. Concluding remarks and discussion of future work are presented in Section 5.

II. Conceptual Model-PASOAC

A. Specification of Conceptual Model

In this section, we specify the PASOAC conceptual model by using the notation of **Entity-Relationship (E-R) Diagram**. In Fig. 2, rectangles represent elements and diamonds represent relationships. In Fig. 2, we define Role (**R**) as a type of user that requires to access the operations (**Op**) of BP. Resource type (**ReT**) is defined as a type of resource that can provide support on the operation (**Op**). Their associated relationships, access and support, are all many-to-many (See two blue arrows in Fig. 2). PASOAC can be formally described as follows,

Definition 1: PASOAC is a tuple $\mathcal{N}$*=(**R**, **Op**, **ReT**, **OPA**, **SPA**, **assigned_op**, **assigned_ret**)*

- ***R**, **Op**, **ReT**, are a set of elements representing Role, Operation, and Resource Type.*
- **OPA** $\subseteq$***Op***$\times$***R**, a relation to map operation to role.*
- **SPA** $\subseteq$***Op***$\times$***ReT**, a relation to map operation to resource type.*
- ***assigned_op:(**r**:R)** $\rightarrow 2^{Op}$, the mapping of role r onto a set of operations. Formally, assigned_op(r)=$\{op \in Op \mid (op, r) \in OPA\}$.*
- ***assigned_ret:(**ret**:ReT)** $\rightarrow 2^{Op}$, the mapping of resource type ret onto a set of operations. Formally, assigned_ret(ret)=$\{op \in Op \mid (op, ret) \in SPA\}$.*

In Fig. 2, two types of authorization policies are also defined within PASOAC as constraints to restrict the use of role access and resource type support during the execution of business process, that are described in the following sections.

B. Constraint I: Synchronization

A sequence of role accesses on operations of BP can be called as *role-flow*, where roles are continuously used to access the operations based on the business logic of process. A sequence of resource type support on operations of BP can be called as *resource type-flow*, where resource types representing the groups of resources provide successional support on operations during the execution of business process. An authorization flow consists of role-flow and resource type-flow, and must be executed consistently with the control flow of BP. This synchronization means that the role access and the resource type support on specific operations can be enabled when the operation starts to execute, and their permission to access and support on the operation is revoked after the operation finishes. It can guarantee that the access and the support are available when they are needed, and no extra permission can be granted to users and resources. Extra permission may cause additional access and support on the operation that may lead to security issues.

In Fig. 2, three red arrows are used to represent the sequence of role access (role-flow), business logic of BP (control-flow), and the sequence of resource type support (resource type-flow) respectively. Authorization synchronization policy is divided into two types, (1) Role Synchronization Policy and (2) Resource Type Synchronization Policy. Role Synchronization Policy is used to restrict the synchronization between the role-flow and control-flow; while Resource Type Synchronization Policy is used to restrict the synchronization between the control-flow and resource type-flow. These two policies are described as follows.

Constraint 1: Role Synchronization Policy*: Let* ***OP*** *be a set of Operations, and* ***R*** *be a set of roles. Let* ***F*** *be a set of execution sequences between operations, Op→Op∈**F**. We say that the role-flow is consistent with control-flow, if there exist operations op_i and op_j, whose execution sequence can be recorded as $op_i \rightarrow op_j \in \boldsymbol{F}$, then all roles that are mapped to op_i and all roles that are mapped to op_j must access the operations op_i and op_j based on the same execution sequence $op_i \rightarrow op_j \in \boldsymbol{F}$.*

In role synchronization policy, by setting up the sequence of role access to be consistent with the execution sequence of the mapped operations, the synchronization between the role-flow and the control-flow of operations is achieved. The role can not access the operation unless it starts to execute. When the execution of the operation finishes, the role access on the operation is revoked. For example, the `Guarantor` can start to access the operation *Guarantor Confirmation* when the operation can start to execute. When the operation *Guarantor Confirmation* finishes, the permission assigned to `Guarantor` to access the operation should be revoked immediately.

Constraint 2: Resource Type Synchronization Policy*: Let* ***OP*** *be a set of Operations, and* ***ReT*** *be a set of resource types. Let* ***F*** *be a set of execution sequences between operations, Op→Op∈**F**. We say that the resource type-flow is consistent with control-flow, if there exist operations op_i and op_j, whose execution sequence can be recorded as $op_i \rightarrow op_j \in \boldsymbol{F}$, then all resource types that are mapped to op_i and all resource types that are mapped to op_j must support the operations op_i and op_j based on the same execution sequence $op_i \rightarrow op_j \in \boldsymbol{F}$.*

Resource type synchronization policy is used to synchronize the sequence of resource type support with the execution sequence of operations. It requires that the resource type can support the operation only when it is the operation's turn to execute. When the execution of the operation finishes, the support of the resource type on the operation is revoked. For example, when the operation of *Lease Application* starts, the `Funder` should be available in terms of securing financial support amount. After the operation finishes, the support from `Funder` should be revoked immediately to avoid the variation of the financial support amount made by the `Funder`.

C. Constraint II: Dependence

Authorization Dependence Policies restrict that a role or a resource type is not able to access or support the operations, until the other role or the other resource type has already accessed or supported specific operations. Authorization Dependence Policies are separated into 5 categories, (1) between roles, (2) between resource types, (3) between roles and resource types, (4) between resource types and roles, and (5) between groups of roles and resource types (See Fig. 2). Here below, the 5 types of dependency policies are defined as follows,

Constraint 3: Role Dependency Policy **($\mathcal{C}_{r\rightarrow r}$)**: *Let* ***OP*** *be a set of Operations, and* ***R*** *be a set of roles. OPA⊆**R**×**OP** is a set of relations of assigned roles on operations. A role dependency policy $\mathcal{C}_{r\rightarrow r}$ can be written as $opa_a \rightarrow opa_b$, where opa_a and opa_b ∈OPA, and opa_b depends on opa_a, i.e. without opa_a, opa_b can not be used in role-flow.*

Role dependency policy ($\mathcal{C}_{r\rightarrow r}$) is used to restrict that the role access on specific operation depends on another role access in role-flow. This is different from role synchronization policy which is used to restrict the sequence of role accesses consistent with the execution sequence of operations. Formally, when $r_i \times op_i = opa_a$, and $r_j \times op_j = opa_b$, only after opa_a has been used in role-flow, then opa_b can be used. For example, in motivating scenario, the access of the `guarantor` on the operation of *Monthly Bill* depends on the access of the `guarantor` on the operation of *Lease Application*. It means that the `guarantor` can repay the rental on behalf of `customer` only if the *lease* *application* is also submitted by the `guarantor`. A role dependency policy can be written as **`Guarantor`**$\times$***Lease Application***$\rightarrow$**`Guarantor`**$\times$***Monthly Bill***.

***Constraint** 4: **Resource Type Dependency Policy** ($\mathcal{C}_{ret\rightarrow ret}$)**:** Let **OP** be a set of Operations, and **ReT** be a set of resource types. SPA$\subseteq$**ReT**$\times$**OP** is a set of relations of assigned resource types on operations. A resource types dependency policy $\mathcal{C}_{ret\rightarrow ret}$ can be written as $spa_a \rightarrow spa_b$, where spa_a and spa_b $\in$SPA, and spa_b depends on spa_a, i.e. without spa_a, spa_b can not be used in resource type-flow.*

Resource type dependency policy ($\mathcal{C}_{ret\rightarrow ret}$) bears the same semantics like role dependency policy ($\mathcal{C}_{r\rightarrow r}$). It is used to restrict the support of a resource type on specific operation to depend on another support of resource type within resource type-flow. This is also different from resource type synchronization policy which is used to restrict the sequence of resource type support consistent with the execution sequence of the operations. Formally, when $ret_i \times op_i$=spa_a, and $ret_j \times op_j$=spa_b, only spa_a has been used in resource type-flow, then spa_b can be used. For example, in motivating example, the support of `lease agent` on the operation of *Monthly Bill* depends on the support of the `lease agent` on the operation of *Lease Application*. It means that `lease agent` can be used to issue bill only the `lease agent` is used to support the lease application. A resource type dependency policy can be written as **`Lease Agent`**$\times$***Lease Application***$\rightarrow$**`Lease Agent`**$\times$***Monthly Bill***.

***Constraint** 5: **Role Resource Type Dependency Policy** ($\mathcal{C}_{r\rightarrow ret}$)**:** Let **OP** be a set of Operations, **R** be a set of roles, and **ReT** be a set of resource types. **R**$\times$**OP**$\subseteq$OPA is a set of relations of assigned roles on operations, and **ReT**$\times$**OP**$\subseteq$SPA is a set of relations of assigned resource types on operations. A role resource type dependency policy $\mathcal{C}_{r\rightarrow ret}$ can be written as $opa_a \rightarrow spa_a$, where opa_a $\in$OPA and spa_a $\in$SPA, and spa_a depends on opa_a, i.e. without opa_a, spa_a can not be used in resource type-flow.*

Role resource type dependency policy ($\mathcal{C}_{r\rightarrow ret}$) is used to restrict a support of a resource type on specific operation to depend on a role access in role-flow. When $ret_i \times op_i$=spa_a, and $r_j \times op_j$=opa_a, Only after the role r_j has been used to access the specific operation op_j, the resource type ret_i can then support the operation op_i. For example, in motivating scenario, the support of `credit assessor` on the operation of *Lease Confirmation* depends on the access of the `guarantor` on the operation of the *Guarantor Confirmation*. It means that, the `credit assessor` can evaluate the credit history of `customer` and `guarantor` only after the `guarantor` has confirmed its information. A role resource type dependency policy can be written as **`Guarantor`**$\times$***Guarantor Confirmation***$\rightarrow$**`Credit Assessor`**$\times$***Lease Confirmation***.

***Constraint** 6: **Resource Type Role Dependency Policy** ($\mathcal{C}_{ret\rightarrow r}$)**:** Let **OP** be a set of Operations, **R** be a set of roles, and **ReT** be a set of resource types. **R**$\times$**OP**$\subseteq$OPA is a set of relations of assigned roles on operations, and **ReT**$\times$**OP**$\subseteq$SPA is a set of relations of assigned resource types on operations. A resource type role dependency policy $\mathcal{C}_{ret\rightarrow r}$ can be written as $spa_a \rightarrow opa_a$, where opa_a $\in$OPA and spa_a $\in$SPA, and opa_a depends on spa_a, i.e. without spa_a, opa_a can not be used in role-flow.*

Resource type role dependency policy ($\mathcal{C}_{ret\rightarrow r}$) is a reverse policy of role resource type dependency policy, in which it enables the role access on specific operation to depend on the support of a resource type. When $ret_i \times op_i$=spa_a, and $r_j \times op_j$=opa_a, only after the resource type ret_i has been used to support the specific operation op_i, the role r_j can then access the operation op_j. In motivating scenario, the access of `guarantor` on the operation *Monthly Bill* depends on the support of `funder` on the operation *Monthly Bill*. It means that the operation *Monthly Bill* can be accessed by the `guarantor`, only after the bill has been issued by the `funder`, rather than `lease agent`. A resource type role dependency policy can be written as **`Funder`**$\times$***Monthly Bill***$\rightarrow$**`Guarantor`**$\times$***Monthly Bill***.

All dependency policies defined until now are used to restrict role access and resource type support from single element's point of view, e.g., a role access depends on another role access. However, they are lack of capability to describe the dependency restriction between the groups of role accesses and resource type supports, where a role and a resource type are grouped and their access and support depends on the access and support of another grouped role and resource type.

***Constraint** 7: **Group of Role and Resource Type Dependency Policy**($\mathcal{C}_{r\times ret\rightarrow r\times ret}$)**:** Let **OP** be a set of Operations, **R** be a set of roles, and **ReT** be a set of resource types. **R**$\times$**OP**$\subseteq$OPA is a set of relations of assigned roles on operations, and **ReT**$\times$**OP**$\subseteq$SPA is a set of relations of assigned resource types on operations. A group of role and resource type dependency policy $\mathcal{C}_{r\times ret\rightarrow r\times ret}$ can be written as $(opa_a, spa_a) \rightarrow (opa_b, spa_b)$, where opa_a and opa_b $\in$OPA, spa_a and spa_b $\in$SPA, and the group of opa_a and spa_a depends on the group of opa_b and spa_b, i.e. without both opa_a and spa_a, opa_b and spa_b can not both be used in role-flow and resource type-flow, respectively.*

Group of role and resource type dependency policy ($\mathcal{C}_{r\times ret\rightarrow r\times ret}$) (See Fig. 2) is used to restrict a role access and a resource type support to depend on the access and support of another group of role and resource type. A role and a resource type can be grouped when they are both working on the same operation. When $ret_i \times op_i$=spa_a, $ret_j \times op_j$=spa_b, $r_i \times op_i$=opa_a, and $r_j \times op_j$=opa_b, only the role r_i and the resource type ret_i has both been used to access and support the specific operation op_i, the role r_j and the resource type ret_j can then be both used to access and support the operation op_j. In motivating scenario, the access of `lawyer` and the support of `lease agent` on the operation of *Return* depends on the access of customer and the support of lease agent on the operation of *Lease Application*. It means that, if lawyer and lease agent are together to deal with the *return* of the product, i.e, not the real product supplier and not real product user, then the lease application must be submitted directly by the customer and the product should be provided by the lease agent. The application submitted by the applicant in person can lower the risk of dispute on the product of return based on a belief that, the more the customer involves, the less the dispute can occur. A group of role and resource type dependency policy can be written as <**Customer**×***Lease Application*****, Lease Agent**×***Lease Application***>→<**Lawyer**×***Return*****, Lease Agent**×***Return***>.

III. Specification of SOAC-Net

In this section, we develop a process model named as SOAC-Net to represent the authorization flow, where the two types of authorization policies defined in last section can be enforced. SOAC-Net is divided into three parts, role-flow, resource type-flow, and constraint-flow. In SOAC-Net, a role-flow and a resource type-flow are derived from the execution sequence of the operations and associated authorization mappings, i.e. the mappings between roles and operations, and the mappings between resource types and operations. These two types of flows are tightly synchronized with the execution sequence (control flow) of the operations. But they are not the same as the control flow since they have capability to manage the sequence of role accesses and the sequence of resource type supports within one operation, where the control flow can only be used to coordinate the execution sequence between operations. Constraint-Flow is used to represent the various authorization dependency policies based on the role-flow and resource type-flow.

A role-flow is defined as a sequence of role accesses on specific operations of BP and is constructed based on two steps (See Fig. 3).

Step 1 The roles accesses on the operations is sequenced based on the operation execution sequence, where the roles that can access the same operation are put together. The role access on operation can be obtained from the given authorization mappings, i.e. the mapping between the role and the operation (See Line 5 in Algorithm 1).

Step 2 If the relationship of the roles put together for accessing the same operation is *exclusive choice*, then *role or-split* and *role or-join* are used to set up these roles accesses on the operation into different branches. One of the branches will be selected during the role-flow execution. On the other hand, if multiple roles need to concurrently access the same operation, *role and-split* and *role and-join* are used to divide these role accesses on the operation into different branches that execute concurrently. These four operators can be used as nested structure. By using this construction method, a role-flow is developed that can not only synchronize the execution sequence of the operations, but can also manage the role accesses within one operation (See Line 7 in Algorithm 1).

Algorithm 1 Algorithm for Constructing Role-Flow

```
Input: R, OP, OPA⊆OP×R, F⊆OP→OP
Output: E⊆OPA→OPA
   {Based on the operation's execution sequence F, and
   the mapping OPA between Role R and Operation OP,
   a sequence of role accesses E is constructed}
Steps:
1:  while NOT the end of F do
2:    /*Check if the operation $op_i$ is mapped with roles by
      using the inverse function of assigned_op(r∈R)*/
3:    if assigned_op$^{-1}$($op_i$ ∈OP)$\neq \emptyset$ then
4:      /*Select all roles that are mapped with $op_i$ into R$'$,
        a subset of Role R. */
5:      $\exists$ R$'$⊆R, $\forall r_i$∈R$'$, $r_i \times op_i$=OPA$'$⊆OPA, $\forall r_j$∈R-
        R$'$, $r_j \times op_i \not\subseteq$OPA;
6:      /*Relationship() is a recursive function on finding
        all relationships of Exclusive-Choice or Concur-
        rency between different OPAs.*/
7:      W=Relationship(OPA$'$);
8:      E=Sequence(E,R$'$,W); /* Based on the set of role
        R$'$ and the set of relationships W, the roles mapped
        with $op_i$ are added in the role-flow E with the
        relationships within OPA$'$. */
9:    end if
10:   Next($op_i$∈ OP); /*Select the next operation F*/
11: end while
```

In role-flow, the operation is used to pass from one role access to the other depending on which operation should be accessed by the specific role. The synchronization between role-flow and control-flow of the operations can then be ensured at runtime. Here below, we present an algorithm to construct role-flow (See Algorithms 1).

A resource type-flow is defined as a set of resource type supports on specific operations, and its construction method is similar to the method of constructing a role-flow. Hence, we will not describe the construction method for resource type-flow in details. In resource type-flow, the operation is also used to pass from one resource type supports to the other depending on which operation should be supported by the specific resource type. This can ensure the synchronization between resource type-flow and control-flow of the operations at runtime. Due to space limit, we ignore the algorithm to construct resource type-flow, which is similar with the algorithm to construct role-flow.

Constraint-flow is constructed based on the authorization dependency policies. A constraint node is used to link role access or resource type support according to the specific

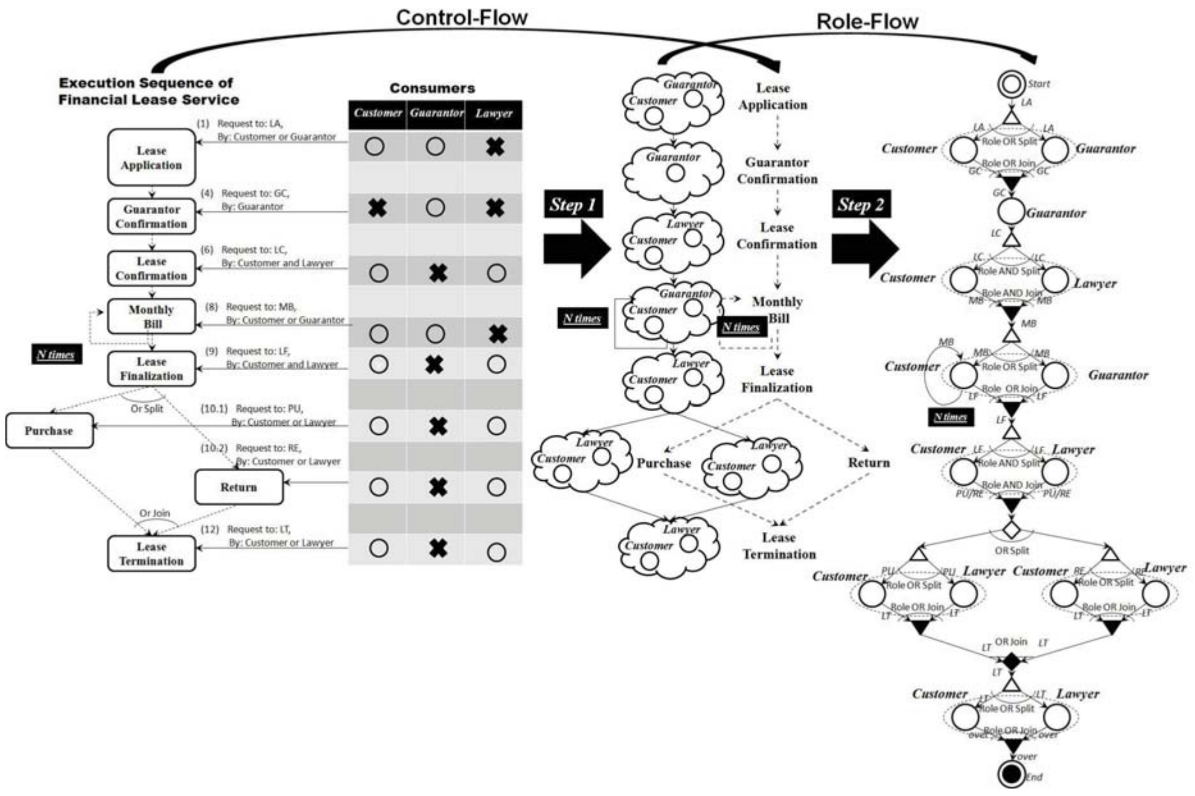
Control-Flow
Role-Flow
Execution Sequence of Financial Lease Service
Consumers
Customer
Guarantor
Lawyer
Lease Application
Guarantor Confirmation
Lease Confirmation
Monthly Bill
Lease Finalization
Purchase
Return
Lease Termination
N times
Or Split
Or Join
(1) Request to: LA, By: Customer or Guarantor
(4) Request to: GC, By: Guarantor
(6) Request to: LC, By: Customer and Lawyer
(8) Request to: MB, By: Customer or Guarantor
(9) Request to: LF, By: Customer and Lawyer
(10.1) Request to: PU, By: Customer or Lawyer
(10.2) Request to: RE, By: Customer or Lawyer
(12) Request to: LT, By: Customer or Lawyer
Step 1
Step 2
Start
End
Role OR Split
Role OR Join
Role AND Split
Role AND Join
OR Split
OR Join

Figure 3. Construction of Role-Flow

dependency policy that the constraint node represents. The ingoing link of the constraint node comes from the depended role access or resource type support; while the outgoing link of the constraint node points to the depending ones. When the depended role access or resource type support finishes dealing with specific operation, the constraint node will be activated by the ingoing link. Through the outgoing link of the constant node, the depending role access or resource type support can be available on specific operations when they are needed. Otherwise, they are not available.

In Fig. 4, we illustrate an example of SOAC-Net based on the scenario in motivating example section. We use white circles to represent role accesses in role-flow, and grey circles with horizon strips to represent resource type supports in resource type-flow. The blue circles with vertical strips are used to represent various types of dependency policies in constraint-flow. The operations are transferred within the SOAC-Net, when they are accessed or supported by specific role or resource type in role-flow and resource type-flow. The authorization synchronization policies are enforced when the operations are transferred within role-flow and resource type-flow. All authorization dependency policies in motivating example are explicitly illustrated in Fig. 4 as constraint-flow. For example, for role dependency policy **`Guarantor`**×***Lease Application***→**`Guarantor`**×***Monthly Bill***, a constraint node is used to link the `guarantor` access on the operation of *Lease Application* with the `guarantor` access on the operation of *Monthly Bill* (See Fig. 4). In that case, the node in role-flow that represents the `guarantor` access on the operation *Monthly Bill* is restricted by an extra ingoing link from the constraint node, where without the activation of this ingoing link, the `guarantor` access on the operation *Monthly Bill* can not be enabled.

IV. Related Work

Role based access control [2], [5], [6] is a widely accepted approach on business process authorization. In RBAC, users acquire permissions through their roles to access the tasks in workflow, rather than they are assigned permissions directly. However, traditional RBAC models in process environment deal with authorization of resources which belong to an individual organization. In web service paradigm, the resources normally spread over multiple organizations. Traditional RBAC models can not be used directly as ready solutions for authorization of BP. We will overview some representative work in the follows.

An access control model CWS-RBAC was proposed in

[3], where a global role is assigned to users to access a sequence of operations, and a local role mapped from global role is assigned to user to access the resources. The authors in [4] propose another concept-*Role Composition* where global role and local role are composed together. If the user is assigned with a global role, then it automatically bears the permissions of the bound local role on resource. In these approaches, the "role" as a concept is part of internal security policy within a BP or a resource, and can not be identified by others. Actually, the BP can only perceive the permissions of resources based on credentials. The mapping of the global role issued from BP with the local role generated in resource is not realistic. In our proposed approach, we introduce resource type (**ReT**) to express characteristics and requirements of resources that are defined by BP, rather than the resource.

In [7], the authors proposed a workflow authorization model (WAM) and an authorization template (AT) to realize the synchronization of role-flow with workflow. Authors in [8] extends the above WAM by using colored petri-net to enforce more policies, e.g. separation of duty (SoD) and binding of duty (BoD). In [9], the authors propose a constrained workflow systems where local and global cardinality constraints as well as SoD and BoD are enforced. However, all above authorization models in workflow environment do not take resource into account, and can not be used in web service environment. The authorization dependency policies are also missing in the existing models.

V. Conclusion and Future Work

In this paper, we propose a business process authorization model PASOAC to illustrate how user access and resource support on business process is managed. Two types of authorization constraints, synchronization and dependency, are identified in PASOAC. A process model SOAC-Net incorporated with PASOAC is presented to ensure that the BP will not be interrupted by authorization issues through enforcing synchronization and dependency policies. In the future, SOAC-Net will be formalized by Petri-Net to facilitate its verification mechanism on examining if authorization policies are defined properly in BP.

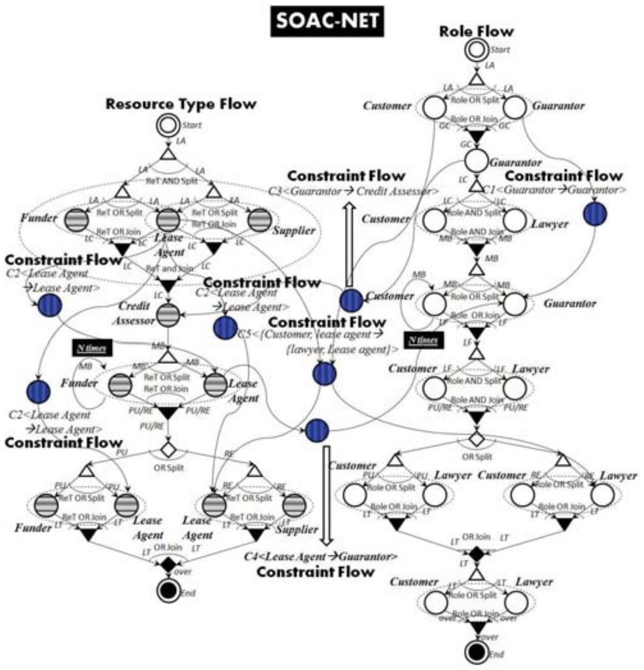
SOAC-NET
Role Flow
Resource Type Flow
Constraint Flow
C3<Guarantor → Credit Assessor>
C1<Guarantor → Guarantor>
C2<Lease Agent → Lease Agent>
C5<{Customer, lease agent → {lawyer, Lease agent}>
C4<Lease Agent → Guarantor>
N times
Customer
Guarantor
Lawyer
Funder
Lease Agent
Supplier
Credit Assessor
Start
End

Figure 4. SOAC-Net

References

[1] Wil M. P. van der Aalst: Verification of Workflow Nets. the 18th International Conference on Application and Theory of Petri Nets.

[2] RS. Sandhu, E. Coyne, H. Feinstein, C. Youman.: Role-based Access Control Models. IEEE Computer, **29**(2), 38–47, (1996).

[3] R. Wonohoesodo, and Z. Tari.: A Role Based Access Control for Web Services. in Proceedings of SCC, 49–56, (2004).

[4] J. Fischer, and R. Majumdar.: A Theory of Role Composition. in Proceedings of ICWS, 49–56, (2008).

[5] D. Ferraiolo, R. Sandhu et al.: Proposed NIST Standard for Role-Based Access Control. ACM Trans. on Information and System Security (TISSEC), **4**(3), 224–274, (2001).

[6] G. Ahn and R. Sandhu.: Role-Based Authorization Constraints Specification. ACM Transactions on Information and System Security (TISSEC), **3**(4), 207-226, (2000).

[7] Vijayalakshmi Atluri and Wei-Kuang Huang.: An authorization model for workflows . in Proc. of the 4th European Symposium on Research in Computer Security (ESORICS'96), (1996).

[8] Zhang Yi, Zhang Yong and Wang Weinong.: Modeling and Analyzing of Workflow Authorization Management. Journal of Network and Systems Management, 12(4), 507–535, (2004).

[9] Tan, K., Crampton, J, and Gunter, C.A.: The consistency of task-based authorization constraints in workflow. 17th IEEE Workshop of Computer Security Foundations, (2004).