Selecting Web Services and Participants for Enforcing Workflow Access Control

San-Yih Hwang, Chuan Yin, Chien-Hsiang Lee
Department of Information Management, National Sun Yat-sen University
Kaohsiung, Taiwan 80424
syhwang@mis.nsysu.edu.tw

Abstract
Web services have emerged as a de facto standard for encapsulating services within or across organization boundaries. Various proposals have been made to compose Web services into workflow so as to meet the goal previously unaccomplished by a single entity. This paper focuses on the workflow access control problem and proposes a Web service selection approach that dynamically chooses a performer for each task in the workflow, not only to satisfy all access control constraints currently but also to increase the chance of completing the entire process in the future. The proposed approach is evaluated using synthetic data and is shown to result in the execution that is less likely to violate any specified access control constraints.

1. Introduction

As SOA emerges as a novel and promising way for future software development, organizations start to encapsulate their services using standards like Web Services Description Language (WSDL)[5], Simple Object Access Protocol (SOAP)[8], and Universal Description, Discovery and Integration (UDDI) [6] so as to interoperate with applications within or across organization boundaries. To complete complicated missions interleaved with humans and information systems, a workflow may involve manual tasks, automatic tasks, and other components. A manual task is completed by a human participant inside or outside an organization, and an automatic task typically invokes a web service (WS) via a predefined interface described by WSDL, which may be prepared by another department or an external service provider.

However, for a given task, there exist a number of humans or WSs that are qualified to perform it. Access control in the workflow specifies the qualification of the performers of various constituent tasks, which could be based on the intrinsic properties of the performers, e.g., the ages, the expertise, the positions, and the roles, or the context, e.g., the current time, the locations of performers, and execution history of the workflow. As a workflow may have to flow through information systems of different organizations, access control constraints are deemed necessary in order to reflect the business rules of the various participating organizations. An access control mechanism on WSs and workflows has to be designed to incorporate the access control definitions and enforce them during workflow execution.

When a task involved in a workflow is ready for execution according to control flow logics described in the workflow process definition, the workflow engine selects a performer for it from several WSs or human participants that are capable of offering the corresponding function. Before such a selection can be performed, access control constraints should be consulted so that WSs or human participants violating access control constraints are filtered out before workflow engine performs the selection. Therefore, a flexible model is required to describe the access control constraints originating from diverse business policies or security concerns. In addition, an algorithm for dynamically determine a performer for each task while maximizing the chance of satisfying all constraints in a workflow is desired.

Consider the following Purchase Process example. In an organization, three employees, P1, P2, and P3 work under a manager P4. Note that a manger is also an employee, and all employees have the rights to start the Purchase Process, which consists of six sequential tasks: Prepare Order ($t_1$), Examine Order ($t_2$), Submit Order ($t_3$), Send Order to Supplier ($t_4$), Perform Payment to Supplier ($t_5$), and Inspect Goods ($t_6$). Tasks $t_4$ and $t_5$ are automatic tasks, whereas the others are manual ones. The entire workflow and the associated access control constraints (which will be described later) are depicted in Figure 1. Figure 2 and 3 show an example role hierarchy and two WSs used by the workflow respectively.

Suppose an employee P1 wants to purchase an item, she first supplies the required information and initiates an instance of Purchase Process. The
workflow management system identifies the employee and designates her as the performer of \( t_1 \) (Prepares Order). When P1 completes \( t_1 \), the “Seniority” constraint bounding \( t_1 \) and \( t_2 \) takes effect and her manager P4 is assigned to perform \( t_2 \) (Examine Order). Furthermore, there is a “Separation of Duties” (SoD) constraint set between \( t_1 \) and \( t_3 \) (Submits Order), which dictates that the participant who performed \( t_1 \) cannot perform \( t_3 \) in the workflow instance[3]. Thus, either P2 or P3 can be selected to perform \( t_3 \). However, it is found that P2 and supplier A are related and therefore are not allowed to execute the same workflow instance, as marked “\((P2,t_3, A,t_4)\) disallowed” in Figure 1. Thus, it is wiser to choose P3 for executing \( t_3 \) because such a selection allows either supplier A or B to execute \( t_4 \) and \( t_5 \), thereby increasing the chance of successful completion of the entire process.

This paper is organized as follows. Section 2 reviews the previous work in workflow access control. Section 3 presents some basic concepts and defines the problem we wish to solve. Section 4 describes our proposed approach, which is then evaluated using synthetic data as reported in Section 5. Finally, Section 6 summarizes the paper and gives future research directions.

2. Related work

For a human participant or a WS to perform a task of a workflow, the workflow management system has to recognize and give authorization to the performer. The authorization must conform to the binding between tasks and performers specified in the process definition. In recent years, the problem of authorizing users to perform tasks within a workflow while enforcing access control constraints have attracted much interest, especially from security community. Bertino et al. [2] proposed methods to specify authorization information, articulate authorization constraints, and attach such information to WS-BPEL process definition. In [13] a method to specify and enforce access control on manual tasks with existing workflow management systems was proposed. A language based on pure-past linear temporal logic was proposed in [12] to specify access control constraints related to past histories of service invocations and role hierarchies. In the survey of [14], several access control models including access matrix model, role-based access control model, task-based access control model, team-based access control model, spatial access control model, and context-aware access control model were introduced.

Access control constraints are rules confining the performers of tasks in accordance to security policies. One of the well-known constraints is separation of duties, which requires mutually exclusive roles to complete a sensitive task [11]. Because security-relevant contextual information can capture the dynamically changing environment, these

![Figure 1. An example purchase process and its access control constraints](image1)

![Figure 2. An example role hierarchy](image2)

![Figure 3. Two WSs conducted by Supplier A and B](image3)
information directly affects the effectiveness of access control [4]. Access control constraints can be classified into context-free constraints and context-sensitive constraints. A context-free constraint on a task determines the set of performers of the task based on static attributes of each performer. The used attributes may include roles, ages, gender, reliability, and responsiveness, whose values generally do not change during the execution period of a workflow instance. An example context-free constraint is the service restriction constraint proposed in [3] that states a certain service cannot be accessed by users under a certain age. A context-sensitive constraint determines the set of performers based on the context of the running workflow instance, which may include attributes of each performer as well as the performers that have performed previous tasks in the same workflow instance. There are quite a few types of context-sensitive constraints that have been proposed. History-based constraints, a major type of context-sensitive constraints that are based on past history of execution, include separation of duties, binding of duties, session limit, and pre-requisite role [1, 3, 4, 11, 12].

Most of the previous work in workflow access control focuses on the specification of various types of constraints and the selection of a performer for each task so as to satisfy all given constraints. Our work differs from previous work in that we select a performer for a task not only to satisfy all the constraints currently but also to increase the chance of satisfying all constraints in the future.

3. Fundamental concepts and problem definition

We formally define the workflows and the two types of task performers, namely participants and WSs, using finite state machine (FSM). Figures derived from the Purchase Process example as mentioned in Section 1 will be used to illustrate these concepts.

**Definition 1: (Workflow)**

A Workflow $WF$ is a FSM that prescribes all possible executions of constituent tasks. Formally, $WF$ is a tuple $(\Sigma, S, s_0, \delta, F)$, where

- $\Sigma$ is a set of tasks,
- $S$ is a finite set of states,
- $s_0$ is a state in $S$, representing the initial state of $WF$,
- $\delta: S \times \Sigma \rightarrow S$ is the transition function of the FSM, which is a partial function that returns the new state resulting from a task in $\Sigma$ being performed when the $WF$ is in some given state, and

- $F \subseteq S$, representing the set of final states, i.e., the states where the interactions with $WF$ can be terminated.

**Figure 4. A Purchase Process represented as an FSM**

Figure 4 illustrates the FSM of the Purchase Process in Figure 1, which contains a sequence of six tasks. When a task is performed, the state transits to the next one. For example, after the task “Employee Prepares Order” is performed, the state transits to the next one, waiting for the arrival of the next task: “Manager Examines Order.”

**Definition 2: (Component WS)**

A component WS is an entity that consists of a workflow of operations prescribing the legal execution orders of these operations.

**Definition 3: (Participant)**

A participant is a person that involves a workflow of activities. Here an activity is defined as a pair of task and role $(t, r)$, meaning the completion of a task $t$ by the person who played the role $r$.

Both component WSs and participants can be described using FSM. Following the example shown in Figure 1, we depict the two component WSs provided by two suppliers A and B (for executing $t_5$ and $t_6$), shown in Figure 5. Figure 6 shows the FSMs for the participants P1, P2, P3, and P4. Note that P4 can play two roles: employee and manager, and is thus capable of performing both “Prepares Order” task (when playing the role of employee) and “Examines Order” task (when playing the role of manager). Here the states that are double circles represent final states. At the end of execution, the workflow of each
component WS and participant must be in either the start state or the final state.

**Definition 4: (Target Workflow)**
A target workflow is a workflow whose tasks are interfaces to operations of component WSs or activities of human participants. Note that the performer of a task can be either a component WS or a human participant.

**Figure 5. The FSMs of two component WSs provided by supplier A and B**

**Figure 6. FSMs of four participants**

**Definition 5: (Authority)**
An authority A is a mapping from a task in the target workflow to a set of performers. Formally, $A: T \rightarrow 2^E$, where $T$ is a set of tasks and $E$ is a set of performers.

The authority of a task $t$ describes the qualification of performers eligible for executing $t$. For example, the authority of $t_1$ as described in Figure 1 is $\{(P1, Emp), (P2, Emp), (P3, Emp), (P4, Emp)\}$, meaning that $P1$, $P2$, $P3$, and $P4$ can all perform $t_1$ when playing the role of Employee.

**Figure 4. Participant FSM for Employee $p1$, $p2$ and $p3$**

**Problem definition**
Given a set of access control constraints as well as a target workflow, a set of component WSs, and a set of participants, how do we dynamically assign a performer for each task in the target workflow so as to increase the chance of successfully completing the workflow instance with all constraints being satisfied?

$t_1$, denoted $\text{SoD}(t_1, t_3)$. More constraints will be elaborated in Section 4.

**Definition 7: (Composition)**
Given a target workflow $W$, a set of component WSs $T$, and a set of participants $H$, the composition of $W$, using $T$ and $H$, denoted $\text{Comp}(W, T, H)$, is an FSM that prescribes all possible delegations.

To illustrate the concept of composition, we use the target workflow, the component WSs, and the participants depicted in Figure 4, 5, and 6 respectively to construct the composition, which is partially shown in Figure 7. Each node in the composition represents a state of the system and is called a configuration.

**Figure 7. Partial composition of $W$ in Figure 4, using $T$ and $H$ shown in Figure 5 and 6 respectively, where the dotted paths violate the constraints**

Note it is possible that some executions in the composition violate a certain access control constraints. In Figure 7, the executions marked in dashed lines violate some of the specified access control constraints. For example, on the leftmost path, $(A, t_4)$ cannot follow $(p_2, r_1, t_3)$ due to the constraint “$(p_2, t_3, A, t_4)$ disallowed”. In addition, $(p_1, r_1, t_6)$ cannot follow $(p_2, r_1, t_3)$ due to $\text{BoD}(t_3, t_6)$.

**Problem definition**
Given a set of access control constraints as well as a target workflow, a set of component WSs, and a set of participants, how do we dynamically assign a performer for each task in the target workflow so as to increase the chance of successfully completing the workflow instance with all constraints being satisfied?
4. Our approach

Our approach to solving the dynamic performer selection problem (for enforcing access control) consists of two phases: the design phase and the execution phase. In the design phase, we aim to build a composition by taking as input the definition of the target workflow, a set of component WSs, a set of human participants, and a set of access control constraints. Specifically, we incorporate as many access control constraints as possible into the FSMs of component WSs and human participants. We subsequently construct a composition using the modified FSMs.

In the execution phase, the composition is used by the access control enforcement module attached to the workflow engine. The access control enforcement module determines which component WS or participant to delegate for a given task by consulting the composition and utilizing runtime contextual information, as well as the access control constraints unenforced in the design phase.

Figure 8 shows the procedure of our proposed approach.

4.1. Modeling the access control constraints

We observed that many types of constraints are about the exclusive and inclusive designation of performers. Thus we propose to use exclusive ordered pairs (abbreviated as EOP) and inclusive ordered pairs (abbreviated as IOP) to describe these access control constraints so that the constraints concerned in the current study can be generalized and examined using a uniform algorithm. We use “→” as the operator for an inclusive ordered pair and “→!” for an exclusive ordered pair.

The antecedent and consequent of each operator take the form of (task, role) for human participant or simply (task) for component WS. For example, an inclusive pair \(<(t_2, r_4)\rightarrow(t_1, r_3)\>\) means that task \(t_2\) must be executed by the same participant playing the role of \(r_4\) who had performed task \(t_1\) under the role of \(r_3\). As another example, \(<t_2\rightarrow!t_1>\) indicates that the component WS that had been delegated to execute \(t_1\) cannot be used to execute \(t_2\).

We survey a wide range of access control constraints described in the literature and identify eight types of access control constraints. We will briefly describe each type of constraints and show how it can be expressed using the proposed exclusive/inclusive ordered pairs, if any.

---

**Figure 8. Architecture of the proposed approach**

1. **Separation of Duties (SoD)**
   Separation of Duties (SoD) constraint indicates that some (sensitive) tasks should not be executed by a participant (with some roles) or component WS who has previously performed a specific task in the same workflow instance. We can transform a SoD constraint into a set of exclusive ordered pairs. For example, to disallow two sequential automatic tasks \(t_1\) and \(t_2\) to be executed by the same WS, we simply specify \(<t_2\rightarrow!t_1>\). As another example, suppose that we do not want two sequential manual tasks \(t_1\) and \(t_2\) to be executed by the same participant who plays either role \(r_1\) or \(r_2\). Two exclusive ordered pairs can be specified: \(<(t_2, r_1)\rightarrow!(t_1, r_1)>\) and \(<(t_2, r_2)\rightarrow!(t_1, r_2)>\).

2. **Binding of Duties (BoD)**
   Binding of Duties (BoD) constraint dictates that two tasks must be executed by the same performer. Let \(t_2\)
be executed after \( t_1 \), the BoD constraint on \( t_1 \) and \( t_2 \) can be directly specified using an inclusive ordered pair: \(<t_2 \rightarrow t_1>\).

3. Session limit
Session limit requires that a participant should not play more than \( n \) role(s) in a single running process instance [11]. For now we only consider the situation of \( n=1 \). We can use a set of exclusive ordered pairs on a role to express such a session limit constraint. For example, suppose that a participant \( p \) can play two roles \( r_1 \) and \( r_2 \) and is qualified to execute three tasks \( t_1, t_2, \) and \( t_3 \) in order, but is subject to the session limit constraint. We can specify the following inclusive ordered pairs for \( p \):
\[
<(t_3, r_1) \rightarrow ! (t_2, r_2)>, \quad <(t_2, r_1) \rightarrow ! (t_1, r_2)>, \quad <(t_3, r_2) \rightarrow ! (t_2, r_1)>,
\]
and \(<(t_2, r_2) \rightarrow ! (t_1, r_1)>\).

4. Pre-requisite roles
A pre-requisite roles constraint on roles \((r_1, r_2)\) specifies that a participant can be assigned to a certain role \( r_2 \) only if he/she had played another role \( r_1 \) before [11]. This constraint can be expressed with inclusive ordered pairs. For example, suppose a participant is capable of executing three sequential tasks \( t_1, t_2, \) and \( t_3 \). A pre-requisite roles constraint on roles \((r_1, r_2)\) can be represented by the following inclusive ordered pairs:
\[
<(t_2, r_2) \rightarrow !(t_2, r_1)>, \quad <(t_3, r_2) \rightarrow !(t_1, r_1)>,
\]
and \<(t_2, r_2) \rightarrow !(t_1, r_1)>\).

5. Service restriction
Service restriction constraints are conditions specified on the static properties of the executors [3]. A service restriction constraint on a task \( t \) can be incorporated into the authority of \( t \) and does not require explicit access control.

6. Seniority
A seniority constraint on a pair of tasks \((t_1, t_2)\) requires that \( t_2 \) be executed by a participant who is more senior than that of \( t_1 \) [7]. Seniority constraints can be checked only when a task (e.g., \( t_2 \)) is ready to execute and can be easily enforced by further confining its authority at runtime. It does not require explicit access control.

7. Location/time-based
A location/time-based constraint places a limit on allowed performances of a task based on the locations of performers and/or the current time [13]. Similar to the seniority constraints, location/time-based constraints can be enforced by confining the authority of a task at runtime and does not require explicit access control.

8. Disallowed delegation sequence
This is the most general type of access control constraints. A disallowed delegation sequence contains delegations which are not allowed to co-exist in that order. For example, a disallowed delegation sequence \((W_1, o_1, W_2, o_2, W_3, o_3)\) specifies that it is invalid to delegate \( o_1 \) to \( W_1 \), \( o_2 \) to \( W_2 \), and \( o_3 \) to \( W_3 \) respectively and to execute them in that order. This constraint will make the paths in the composition \((W_1, o_1, W_1, o_2, W_2, o_2, W_2, o_3, W_3, o_3)\) and \((W_1, o_1, W_1, o_2, W_3, o_2, W_3, o_3)\) invalid because both paths contain the subsequence \((W_1, o_1, W_2, o_2, W_3, o_3)\). In fact, we can extend the constructs of inclusive/exclusive ordered pairs to express disallowed delegation sequences of length 2. For example, a disallowed delegation sequence \((W_1, o_1, W_2, o_2)\) can be expressed as a exclusive order pair \(<W_1, o_2 \rightarrow ! W_1, o_1>\). However, disallowed delegation sequences of length greater than 2 can only be checked at runtime. We will describe how to handle disallowed delegation sequences in subsection 4.4.

4.2. Adjusting the FSMs of participants and component WSs

As seen, the first four constraints listed in the last subsection, namely SoD, BoD, Session limit, and Pre-requisite role, can be expressed using inclusive ordered pairs and/or exclusive ordered pairs. We will show that both inclusive ordered pairs and exclusive ordered pairs can be enforced through refining the FSMs of relevant participants or component WSs.

Consider the FSM of a component WS \( W \) that consists of five operations as shown in Figure 9. Suppose that there is constraint \(<t_4 \rightarrow t_1>\) and that \( t_1 \) and \( t_4 \) can be delegated to \( o_1 \) and \( o_2 \) of \( W \) respectively. Among the four possible executions prescribed by the FSM, \((o_1, o_2, o_3)\) violates \(<t_4 \rightarrow t_1>\).

For the FSM of a given participant or component WS, there may be paths that contain transitions violating some access control constraints and should be avoided. To prevent such violation, one can simply eliminate any transition of the path from the FSM. For example, by removing transition \( o_3 \) from the FSM depicted in Figure 10, the path that violates \(<t_4 \rightarrow t_1>\) cease to exist. However, doing so may also disallow some valid execution, e.g., \((o_2, o_3, o_4)\).

Without loss of generality, consider the FSM of a component WS \( W \) shown in Figure 10(a) and an exclusive ordered pair \( t_2 \rightarrow t_1 \), where \( t_1 \) and \( t_2 \) can be delegated to \( o_1 \) and \( o_2 \) of \( W \) respectively. Obviously, there is a path leading from \( o_1 \) to \( o_2 \) that violates \( t_2 \rightarrow t_1 \). We first identify all paths that lead from \( o_1 \) to \( o_2 \). Here there are transitions \( o_1: (B, X), o_2: (A, X), o_2: (A, X), \) and \( o_2: (A, X) \).
(Y, C), and \( o_j: (Y, D) \), where \( i \neq 1 \) and \( j \neq 2 \). We duplicate all states leading from X to Y, and the duplicated states of X and Y are called X’ and Y’ respectively. Then we remove the transition \( o_1: (B, X) \) and construct a new transition \( o_1': (B, X') \). In addition, we build a new transition \( o_j': (Y', D) \). The resultant FSM is shown in Figure 10(b). It can be seen that there is no path leading from \( o_1 \) to \( o_2 \) in the adjusted FSM and that all the valid paths are preserved.

![Figure 9. The FSM of a component WS that contain a path violating \( t_i \rightarrow t_i \), where \( t_i \) and \( t_j \) can be delegated to \( o_i \) and \( o_j \) respectively.](image)

We then consider the component WF shown in Figure 11(a) and an inclusive ordered pair \( <t_2 \rightarrow t_1> \). Obviously, there is a path leading from \( o_1 \) to \( o_2 \), which violates \( <t_2 \rightarrow t_1> \). We can again identify the paths leading from \( o_1 \) to \( o_2 \) and duplicate all states leading from X to Y. We then eliminate the transition \( o_1: (B, X) \) and construct a transition \( o_1': (B, X') \). In addition, we remove the transition \( o_2: (Y, D) \) and construct two new transitions \( o_2: (Y', C) \) and \( o_2: (Y', D) \). It can be seen that the adjusted FSM satisfies \( <t_2 \rightarrow t_1> \) and preserves all valid paths.

![Figure 10. (a) the original FSM (b) the adjusted FSM that satisfies \( <o_2 \rightarrow o_1> \).](image)

The pseudo code for adjusting FSMs of participants and WSs to satisfy all access control expressed using exclusive ordered pairs and inclusive ordered pairs are eliminated here for brevity.

![Figure 11. (a) the original FSM (b) the adjusted FSM that satisfies \( <o_2 \rightarrow o_1> \).](image)

### 4.3. Building the composition of the target FSM using participants and component WSs

Using the adjusted FSMs of participants and component WSs as well as the FSM of the target workflow, we are able to build a composition that includes all legal sequences of delegations using the algorithm described in our previous work [9]. For those exclusive/inclusive ordered pairs specified between operations (activities) of different component WSs (participants), we can apply the same algorithms described in Section 4.2 to further adjust the composition.

We use the aggregated reliability proposed in [9] as a metric to select component WSs and participants. As mentioned, the workflow execution environment is a failure-prone environment. Each transition in the composition indicates a delegation of a task to some operation in a component WS or some activity in a human participant, which has a chance of failure during the execution. The reliability associated with a delegation \( W_i.o_j \), denoted \( R(W_i.o_j) \), indicates the probability that the operation \( o_j \) of component WS \( W_i \) can be successfully executed. In our previous work [9], we model the composition as a Markov chain by treating each configuration as a state. Suppose that there is a delegation \( W_i.o_j \) leading from configuration \( X \) to \( Y \) and \( W_i.o_j \) is used to execute task \( t \) in the target workflow. The transition probability \( P_{XY} \) can be computed as the product of the arrival probability of \( t \) and the probability that \( W_i.o_j \) is selected and successfully executed when the current configuration is \( X \). Thus, the stationary probability of each configuration \( C \) can be computed using power method [10] and is called the aggregated reliability of \( C \) because it indicates the chance that \( C \) will leads to a final configuration in a failure-prone environment.
The computed aggregated reliabilities of configurations can be directly used for selecting delegation. Specifically, when a task \( t \) arrives and the current configuration is \( C \), all delegations incident from \( C \) and capable of executing \( t \) are considered. The delegation that has the highest product of the reliability and the aggregated reliability of the destination configuration will be chosen. This approach is based on aggregated reliability and is called AR for brevity. For detailed description on the computation of aggregated reliabilities and the AR algorithm, please refer to [9].

4.4. Enforcing disallowed delegation sequence at runtime

There may be further access control constraints that cannot be expressed using exclusive/inclusive ordered pairs and are not based on simple runtime attributes. Some of these constraints can be expressed using disallowed delegation sequences as described in Section 4.1. In this subsection, we describe a method to avoid the violation of disallowed delegation sequences at runtime.

To prevent disallowed delegation sequences, we examine the composition FSM and check if there exists a path \( p \) that is \( k \)-step following the current configuration and contains a disallowed delegation sequence, where \( k \) is a user-specified constant. In this case, we set the reliability of the last transition that appears in \( p \) as 0. For example, consider the disallowed delegation sequence \((w_1.o_1, w_1.o_2, w_4.o_4)\) and the FSM shown in Figure 12. Let \( A \) be the current configuration and a task \( t_2 \) arrives, where \( t_2 \) can be delegated to either \( w_1.o_2 \) or \( w_2.o_2 \). By looking ahead \( k (=2) \) steps, we know that by delegating \( w_1.o_2 \) to \( t_2 \), there is high chance to result in an invalid path \((w_1.o_1, w_1.o_2, w_3.o_3, w_4.o_4)\). Thus, we can set the transition probability on \((E, G)\) to be 0 and re-compute the aggregated reliabilities of some affected configurations. Specifically, the aggregated reliabilities of \( E \) and \( C \) will be reduced after the re-computation. Thus the chance of selecting \( w_1.o_2 \) for \( t_2 \) will be lower.

5. Performance evaluation

We used a trip planning (TripPlan) scenario to illustrate the performance of our approach. For simplicity, we consider only automatic tasks in this scenario and regard TripPlan as a target workflow that consists of interfaces to component WSs. The user who initiates TripPlan workflow is entitled to book flight, reserve accommodation, buy travel insurance, and pay for the trip, as depicted in Figure 13. There are seven component WSs as shown in Figure 14.
We consider the following access control constraints:

exclusive/inclusive ordered pairs:

\[ \text{EOP}<\text{Payment} \rightarrow !\text{Insurance}> \]
\[ \text{IOP}<\text{BookHotel} \rightarrow \text{BookFlight}> \]

disallowed delegation sequences:

\[ <w_1o_3,w_6o_5,w_7o_6> \]
\[ <w_1o_4,w_6o_5,w_7o_6> \]

Obviously, the adjusted FSMs of both \( w_4 \) and \( w_5 \) contain no “Book Hotel” so as to preserve IOP, and \( w_6 \) has to be excluded due to its violation of EOP. We called our approach Adjusted AR because it selects the performer of each task based on the aggregated reliabilities of the adjusted composition. Adjusted AR is compared to two other approaches, namely random and AR. The random selection method randomly chooses a qualified performer for each task without checking access control constraints. The AR selection method makes the performer selection based on the aggregated reliabilities of the original (un-adjusted) composition. For the two approaches, access control constraints are consulted only at the end of each execution to determine the success or failure of an execution instance.

To evaluate the performance of our proposed approach, we chose the success rate as the primary performance metric, which measures the ratio of the execution sequences whose tasks are all successfully delegated to and executed by component WSs without violating any access control constraint. For both AR and random methods, at the end of each execution, the specified access control constraints have to be verified. Thus, we also report for the two approaches the denial rate for EOP and IOP and the denial rate for disallowed delegation sequences. The former indicates the ratio of all execution sequences denied by the specified EOP/IOP constraints, while the later measures the ratio of all execution sequences violating some disallowed delegation sequences.

Our experiments ran on a PC with a Intel CoreDuo L2400 CPU(1.66 GHz) and 1GB RAM running Windows XP professional. We randomly generated 10,000 execution sequences of the target workflow and assumed a fixed reliability probability for each operation or activity (called delegation reliability). Figure 15 shows the success rates of each algorithm under various delegation reliabilities. It can be seen that the success rates of Random and Adjust AR gradually increase with the delegation reliability, and Adjusted AR has the highest success rates. AR strategy has the lowest success rates when the delegation reliabilities are greater than 0.7, because the delegation that leads to the largest aggregated reliability tends to violate some access controls, resulting in failed execution. This is justified by the observation shown in Figure 16 that AR has much higher EOP + IOP denial rates when the delegation reliabilities are greater than 0.7. Figure 17 shows that Adjusted AR has the lowest denial rates induced from the violation of disallowed delegation sequences.

Figure 15. Success rates of the three methods across different delegation reliabilities

Figure 16. Denial rates due to the violation of exclusive/inclusive ordered pairs of the three methods
6. Conclusion

This paper investigates the problem of performer selection for each task in a workflow in presence of access control constraints. We proposed to use exclusive/inclusive ordered pairs and disallowed delegation sequences to model many types of access control constraints in the literature. The exclusive/inclusive ordered pairs are used to adjust the FSMs of component WSs and participants before composition. The disallowed delegation sequences are checked at runtime to enable the selection of a better performer for each task in the workflow. The proposed approach is evaluated using synthetic data and is shown to result in the execution that is less likely to violate any specified access control constraints.

The current approach involves the evaluation of the disallowed delegation sequences at runtime, which could be time consuming. We are currently investigating more efficient methods for handling the disallowed delegation sequences. In addition, more experiments are needed to shed the light on the efficiency and other measures of the proposed approach in a more systematic way. Finally, Incorporating access control with QoS measures other than reliability will be explored in the future.

7. References


