A Secure Workflow Model

Patrick C. K. Hung[1] **Kamalakar Karlapalem**[2]

[1]*CSIROMathematicalAndInformationSciences*
GPO Box 664, Canberra, ACT 2601, Australia.
[2]*InternationalInstituteOfInformationTechnology*
Gachibowli, Hyderabad 500019, India.
Email: Patrick.Hung@csiro.au, kamal@iiit.net

Abstract

Workflow Management Systems (WFMSs) are becoming very popular and are being used to support many of the day to day workflows in large organizations. One of the major problems with workflow management systems is that they often use heterogeneous and distributed hardware and software systems to execute a given workflow. This gives rise to decentralized security policies and mechanisms that need to be managed. Since security is an essential and integral part of workflows, the workflow management system has to manage and execute the workflows in a secure way. The prolific use of workflow management systems for critical and strategic applications gives rise to a major concern regarding the threats against integrity, authorization, and availability. In this paper, we propose an authorization model with a set of invariants for workflows from the aspects of agents, events and data, and prove that if they hold, the workflow execution is secure. Further, we develop the authorization model by a multi-layered state machine. The novel part of this model is separating the various aspects of control in a workflow and portraying it as a multi-layered architecture for analyzing the flow of authorizations.

Keywords: Multi-Layered State Machine, Workflow Security, Authorization.

1 Introduction

Many of the complex day to day workflows in most large organizations are facilitated and conducted by Workflow Management Systems (WFMSs). One of the major problems with WFMSs is that they often use heterogeneous and distributed hardware and software systems to execute a given workflow. This gives rise to decentralized security policies and mechanisms that need to be managed. Since security is an essential and integral part of workflows, the WFMS has to manage and execute the workflows in a secure way. In particular, a robust secure workflow model is needed to allow controlled access of data objects, secure execution of tasks, and efficient management and administration of security (Joshi et al. 2001, Kang et al. 2001, Thuraisingham et al. 2001). The prolific use of WFMSs for critical and strategic applications gives rise to a major concern regarding the threats against the security properties as follows:

- *Integrity* prevents the unauthorized modification of information (Hung 2001). Integrity ensures the correctness and appropriateness of the content of a piece of information. Integrity is also related to the legitimate pattern of operations in data accesses. In a workflow, no data should be modified or corrupted by unauthorized agents.
- *Authorization* is the process of identifying to the system the various functions which a user may undertake (Hung 2001). Particular privileges may be associated with certain roles to execute tasks or access certain resources. In a workflow, authorization means that no data or resource is accessed by unauthorized agents at anytime.
- *Availability* prevents the unauthorized withholding of information or resources (Hung 2001). The resource should be available at hand within a specific time frame (session) once it is needed. In a workflow, data or resource(s) must be at the agent's disposal when needed for legitimate use. In particular, a workflow cannot be done completely if any agent is not available during the task execution.

In order to ensure these four security properties of a workflow, a WFMS must manage and protect all the information of a workflow from build-time to run-time within a secure framework. The secure framework has to prevent any unauthorized modification of data and to enforce the legitimate pattern of operations in data accesses by the agent(s) for executing a task. Thus, the WFMS needs to know when and what data and privilege(s) to be granted to the agent(s) and also to be revoked from the agent(s) for executing a task. Further, the WFMS needs to know the conflict of interest (Nyanchama & Osborn 1999, Hung 2002) for every agent and spread the assignment of task(s) and data to agent(s) as wide as possible in order to reduce security risks. The specialized structure of a WFMS makes the security requirements different from other computing systems. In general, security requirements are particularly needed in a workflow when:

- the workflow manipulates confidential or sensitive information.
- the information moves in and out of a set of agents.
- there are authorization procedures for different resources that need to be enforced in the workflow.

In order to comprehend these problems we need to address two important issues, namely, getting the workflows executed to completion, and having a secure workflow. In particular, the security pilferage through the accessing of build-time and run-time data between a WFMS and agents has to be monitored, controlled and reported. Most of the work done in the workflow research area in both academia and industry are concentrated on the infrastructure of WFMS and transaction management in workflow execution. In

this paper, we propose an authorization model with a set of invariants for workflows from the aspects of agents, events and data, and prove that if they hold, the workflow execution is secure. We develop the authorization model by a multi-layered state machine. In the rest of the paper, section 2 describes the related work, section 3 describes the secure workflow model, section 4 discusses the conclusions.

2 Related Work

Security is an essential and integral part of workflows, and it has become an important topic in the research community as well as the industry. Many researchers are working on workflow standards. The Workflow Management Coalition (WfMC) is an organization that focuses on the advancement of workflow management technology in industry. WfMC summarizes a number of security services (WfMC 2001) for a conceptual workflow model that includes authentication, authorization, access control, audit, data privacy, data integrity, non repudiation, security management and administration. Their approach introduces a security profile that identifies the security services to be applied for inter-operability between two parties. Further, WfMC proposes an extension to the inter-operability protocol to support the workflow service authentication data and the workflow interoperability protocol data. Although WfMC presents the protocol for inter-operability between two parties, the inter-operability protocol proposed by the WfMC does not consider the flow of authorizations among parties, tasks and resources during the workflow execution.

Access control is mainly used to tackle the security properties of *Authentication* and *Authorization*. *Discretionary Access Control (DAC)* (Pernul 1992) is used to model the security of objects on the basis of a subject's access privileges. DAC defines what kind of access a subject has to an object, and a set of predicates to represent access rules such as read, write, delete, create and copy. In other words, DAC controls the access from subjects to objects. So far, DAC only applies to control of system-oriented resources like database, file system, etc. In particular, workflow security requires support for per-task granularity for subjects and objects. For example, a graduate student submits a research paper to a conference. It is obvious that the graduate student has the access right "write" to his paper. Once the paper is submitted to the conference, the access right "write" (of that particular copy of the paper) will be temporarily revoked from the graduate student. After the reviewers examine the paper, they may suggest some comments and require the graduate student to revise the paper. Then, the graduate student can have the access right "write" of the paper again. DAC can represent that the research student (subject) has the access right "write" to the paper (object) and the reviewers (subjects) have the access right "read" to the paper (object), but they cannot handle *when* to grant/revoke the access rights of the object to/from the subjects in this case. This is the major motivation for extending the work on the security properties of *Integrity* and *Authorization* for workflow security.

The Workflow Authorization Model (WAM) (Atluri & Huang 1996, Atluri et al. 1997) presents a conceptual, logical and execution model which concentrates on the enforcement of authorization flow in task dependency and transaction processing by using Petri Nets (PN). The workflow designer defines the static parameters of the authorization using an Authorization Template (AT) during the build-time of the workflow. When the task starts execution, the AT is used to derive the actual authorization. In a Multilevel Secure (MLS) workflow environment, tasks are assigned to different security levels. Further, WAM extends the PN model by proposing a Secure Petri Net (SPN) that is used to detect and prevent all the task dependencies that violate security. Though WAM discusses the synchronization of authorization flow with the workflow and specification of temporal constraints in a static approach, it is not sufficient to support workflow security. This is because workflows need a more dynamic approach to synchronize the flow of authorizations during the workflow execution. For example, the privileges will be granted/revoked to/from the agents according to the events generated during the task execution. WAM only concentrates on the authorization in a task's state and primitives, not the authorization in resource accesses. Further, WAM does not discuss the order of operation flow within the same object in a task. In a workflow, we need to investigate the flow of authorizations in different aspects such as tasks, events and resources. Further, we also need to consider the flow of authorizations from the aspect of agents, not only the interdependency among tasks in a workflow. In other words, WAM only supports a static approach for handling the flow of authorizations in the workflow and data layer. WAM only discusses the Authorization Base (AB) to store all the privileges granted during a workflow execution but it does not consider the concept of history from the aspects of agents and tasks. WAM grants all the authorizations to an agent once the task starts execution and it revokes all the authorizations from an agent once the task is completed, but it does not monitor the event(s) generated from the agent during the execution of task. WAM handles the security property of *Authorization* and MLS handles the security property of *Integrity* in the task dependencies, but they do not handle the security property of *Availability*.

Based on the Workflow Authorization Model (WAM), a web-based WFMS called SecureFlow is proposed in (Huang & Atluri 1999). SecureFlow uses a simple 4GL language, such as SQL, to specify authorization constraints. Though SecureFlow supports the security constraints for role assignments, it is not sufficient for workflow security because workflow security needs to support different security constraints from buildtime to run-time of a workflow. SecureFlow handles the security property of *Authorization*. but it does not handle the security properties of *Integrity* and *Availability*.

The flow of authorizations included the inter-tasks or intra-task scenario from the point of agent and the event generated from the point of task are also involved in the workflow security. A secure workflow model must be able to ensure the security properties of *Integrity*, *Authorization* and *Availability*. As a result, our model provides a framework to control and monitor the security properties for a secure workflow model from the aspects of workflow (task), control (event) and data (document). Further, our model considers the history of an agent (i.e., H^A) and task (i.e., H^T) as an important path to identify potential security threat(s).

3 Secure Workflow Model

The information processed in a workflow is highly valued and it is important to protect this information against security threats. The workflow modeler defines a security policy that typically reflects the security requirements for the workflow. A security policy is a set of rules and procedures regulating the use of information including its processing, storage, dis-

tribution and presentation (Hung 2001). The term security policy is used at the organization level and the set of rules and procedures are used to address security requirements. The security requirements are types and levels of protection necessary for equipment, data, information, applications, and facilities to meet a security policy (Hung 2001). For example, a security requirement may be just as simple as "a person may read a document if the clearance of the person is greater than or equal to the classification of the document". Referring to the WfMC (Hollingsworth 1995), a workflow is formally defined as the computerised facilitation or automation of a business process, in whole or part. Based on this definition, we define a secure workflow as follows.

Definition 1: A **secure workflow** is a computer supported business process that is capable to against security threats and further satisfies the security requirements defined by the workflow modeler.

Definition 2: A **secure Workflow Management System (WFMS)** is a workflow management system that can specify, manage and execute a secure workflow.

Formally, a workflow (WF) is represented as a partially ordered set of tasks (T) that is coordinated by a set of events (E). The order of task execution is orchestrated by matching the input and output event(s) of each task. An event can be either a data event or control event. Each task represents a piece of work that needs to be done by an agent (A). Further, each task is given a Temporal Access Control (TAC) specification, which describes the possible legal sequences of document accesses during the task execution. An agent needs to get certain access privileges (PR) (e.g., "read", "write" and "read-write") to a set of documents (D) during the task execution. Let entities of a secure workflow, namely, sets of tasks (T), events (E), agents (A), Temporal Access Controls (TAC), documents (D) and privileges (PR), respectively, be:

- $T = \{t_1, t_2, \ldots, t_m\}$ is the set of m tasks.
- $E = \{e_1, e_2, \ldots, e_t\}$ is the set of t events. A special event $e^{start(workflow)}$ triggers the *first* task of a workflow and another special event $e^{completed(workflow)}$ is generated after the *last* task of a workflow.
- $A = \{a_1, a_2, \ldots, a_n\}$ is the set of n agents.
- $TAC = \{tac_1, tac_2, \ldots, tac_m\}$ is the set of m temporal access controls.
- $D = \{d_1, d_2, \ldots, d_p\}$ is the set of p documents.
- $PR = \{pr_1, pr_2, \ldots, pr_q\}$ is the set of q privileges.

The relationships among these entities are the following:

- $C : A \longmapsto T$ gives a set of tasks that the agent has the abilities to execute. To illustrate, $C(a_i) = \{t_{i_1}, t_{i_2}, \ldots, t_{i_k}\}$ is the set of k tasks that can be executed by the agent a_i, i.e., $C(a_i) \subseteq T$.
- $C^{-1} : T \longmapsto A$ gives a set of agents that has the abilities to execute a task. To illustrate, $C^{-1}(t_i) = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ is the set of k agents that can execute the task t_i, i.e., $C^{-1}(t_i) \subseteq A$.
- $N : T \longmapsto A$ is a one-to-one mapping that gives an agent that is assigned to execute the task. To illustrate, $\mathrm{N}(t_i) = a_i$ is the agent that is assigned to execute the task t_i, i.e., $N(t_i) \in A$.
- $P : D \longmapsto PR$ gives a set of privileges that can be acquired for a document. To illustrate, $P(d_i) = \{pr_{i_1}, pr_{i_2}, \ldots, pr_{i_k}\}$ is the set of k privileges that can be acquired on the document d_i, i.e., $P(d_i) \subseteq PR$.
- $F : T \longmapsto TAC$ is a one-to-one mapping that gives, for each task, t_j, the corresponding TAC, tac_{t_j}, i.e., $F(t_j) \in TAC$.
- $G : TAC \longmapsto D \times PR$ is a mapping that gives the set of document/privilege pairs specified in a TAC. To illustrate, $G(tac_i) = \{(d_{i_1}, pr_{i_1}), (d_{i_2}, pr_{i_2}), \ldots, (d_{i_k}, pr_{i_k})\}$ is the set of k document/privilege pairs specified in the tac_i where $\forall_{j=1,2,\ldots,k}\ pr_{i_j} \in \mathrm{P}(d_{i_j})$.

Note that the composite function $F \circ G : T \longmapsto D \times PR$ gives the set of document accesses that are needed during the execution of a task. Based on the above description for a workflow, every task is assigned to an agent and the flow of tasks is orchestrated by events. Therefore, it is obvious that there may be many legitimate paths for a workflow. This means that some of the tasks may not be executed in a workflow if they are not on the path of execution flow. For every legitimate path, a set of authorizations is needed for executing tasks and accessing documents. Tasks and documents may contain a lot of sensitive information that may cause threats to a workflow.

In this section, we present a secure workflow model using a multi-layered state machine. A state-machine model describes a system as an abstract mathematical state machine with a set of transition functions. A multi-layered state machine describes a system in different layers where each layer is an abstract mathematical state machine with a set of transition functions. The interaction between two mathematical state machines at different layers is triggered by an event. The multi-layered state machine is used to manage and monitor the flow of authorizations at different layers for a secure workflow execution. The novel part of this model is separating the various aspects of control in a workflow and portraying it as a multi-layered architecture for analyzing the flow of authorizations. There are three layers in a secure workflow: *workflow*, *control* and *data*. The major motivations for using a multi-layered state machine are:

- Different aspects of the flow of authorizations can be modeled in a single framework. The workflow layer is a model in which authorizations can be granted to agents only during the execution of the assigned task and will be revoked as soon as the task execution is finished. The control layer is a model in which relevant event(s) can be generated only during the task execution. The data layer is a model in which document authorizations can be granted and revoked from agents based on the occurrence of event(s) during the task execution.
- A multi-layered state machine supports concurrent states. These states trigger one or more lower-layer state machines that are executed in parallel in the context of the upper layered state machine. This is a practical scenario for a secure workflow model because there may be more than one task running concurrently and also an agent may generate a set of events during the task execution. This set of events may trigger a set of document access activities running concurrently.
- From the point of view of authorization administration, the security administrator can apply

different security services to handle the security properties in different layers. For example, the security service *Discretionary Access Control (DAC)* can be applied to the *Workflow Layer* and *Data Layer* to handle the security property of authorization for assigning/revoking tasks and documents to/from agents, respectively.

- A multi-layered state machine can enable the analysis, simulation and validation of the WFMS under study before proceeding to implementation. The state machine has the advantage of visually depicting the flow of authorizations and presenting all properties, relationships and restrictions among states, such as concurrency, synchronization, control flow dependency and temporal relationships. Once a system has been modeled as a net (Michelis 1999), properties of the system may be represented by similar means, and correctness proofs may be built using the methods of net theory.

In the multi-layered state machine, we support a dynamic approach for handling the flow of authorizations in the workflow and data layer by monitoring the event(s) generated from the control layer in a multi-layered state machine. Our model can capture the event(s) generated from the agent during the execution of a task, and it can grant and revoke the authorizations based on the occurrence of event(s). The advantage of this dynamic approach is to avoid an over-privileged agent at anytime during the execution of a task. Further, our workflow model can capture the sequence of actions of the agent to deduce or revise the task's semantics. The task semantics include aspects derived from the coordination of tasks in the workflow, the agent to execute the task and the resources needed involved in the execution of the task. In particular, our model can support the concurrent execution of tasks with relevant data using the characteristics of a state machine (Booch et al. 1999). The advantage of supporting such a concurrent model is to detect and prevent the security threats such as conflict of interest during a workflow execution. For example, a security threat may occur if an agent can access different documents from different tasks concurrently. In some cases, the agent has to release the document for another task to proceed. For example, an agent *a1* executes the task *t1* and another agent *a2* executes the task *t2* concurrently. The agent *a1* needs to write the document *d1* and then the agent *a2* also needs to write the document *d1*. Since there is concurrency in the document access, the agent *a1* needs to release the document *d1* with the privilege "write" before the agent *a2* can get the "write" access to the document *d1*. In the multi-layered state machine, we support the agent *a1* to release the document *d1* by an event during the execution of task *t1*. Then, the agent *a2* can be granted the access privilege "write" to the document *d1* during the execution of task *t2*. Furthermore, our model can show the feasibility of applying different security services to tackle the security properties in different layers.

In specifying a secure workflow model, we first need to define the security-relevant state variables. Our state variables correspond to the entities and some additional variables are needed for the transition functions.

- LT = a set of positive integers Z^+ representing discrete time.
- $I^T(t)$ = an unique identifier for a task t where $t \in$ T, i.e., $I^T(t1) = I^T(t2)$ if and only if $t1 = t2$.
- $I^E(e)$ = an unique identifier for an event e where $e \in$ E, i.e., $I^E(e1) = I^E(e2)$ if and only if $e1 = e2$.
- $I^A(a)$ = an unique identifier for an agent a where $a \in$ A, i.e., $I^A(a1) = I^A(a2)$ if and only if $a1 = a2$.
- $I^{TAC}(tac)$ = an unique identifier for a temporal access control tac where $tac \in$ TAC, i.e., $I^{TAC}(tac1) = I^{TAC}(tac2)$ if and only if $tac1 = tac2$.
- $I^D(d)$ = an unique identifier for a document d where $d \in$ D, i.e., $I^D(d1) = I^D(d2)$ if and only if $d1 = d2$.
- IE(t) = a subset of the sets of events (i.e., a set of input event sets) where each element in IE(t) (i.e., a set of events) can independently trigger the task t, i.e., $t \in$ T and IE(t) $\neq \emptyset$. To illustrate, IE(t) = {{*e1*, *e2*},{*e1*, *e3*}} means that either the set of events {*e1*, *e2*} or {*e1*, *e3*} can trigger the task t starts execution.
- OE(t) = a subset of the sets of events (i.e., a set of output event sets) where only one element in OE(t) (i.e., a set of events) will be generated when the task t is completed, i.e., $t \in$ T and OE(t) $\neq \emptyset$. To illustrate, OE(t) = {{*e1*, *e2*},{*e1*, *e3*}} means that either the set of events {*e1*, *e2*} or {*e1*, *e3*} is generated once the task t is completed successfully.
- DT(t) = a set of dependent tasks of a task t. Thus, DT(t) = {*t1* | $\forall$ *t1* where *se* $\in$ IE(t), *se1* $\in$ OE(*t1*) such that *se* $\bigcap$ *se1* $\neq \emptyset$} where t, *t1* $\in$ T and *se*, *se1* $\subseteq$ E. Note that DT(t) = $\emptyset$ if the task t is the *first* task of a workflow. There exists a combination of output events from the set of dependent tasks DT(t) that satisfy the input events of a task t if and only if, $\forall$ *se* $\in$ IE(t),
  - $\bigcup_{\forall t2 \in DT(t)}$ {e | $e \in$ *se* $\bigcap$ *se1* where *se1* $\in$ OE(*t2*)} = *se*.

  For illustration, here is an example. The task *t1* has two dependent tasks $t2_1$ and $t2_2$. The output and input events of each task is shown as follows:
  IE(*t1*) = {{*e1*, *e2*},{*e1*, *e3*}}
  DT(*t1*) = {$t2_1$, $t2_2$}
  OE($t2_1$) = {{*e1*}}
  OE($t2_2$) = {{*e2*},{*e3*}}
  Thus, there may be two cases that can occur:
  - *Case 1:* If the *se* = {*e1*, *e2*}, then {*e1*} $\bigcup$ {*e2*} = *se* where {*e1*} is generated from task $t2_1$ and {*e2*} is generated from task $t2_2$.
  - *Case 2:* If the *se* = {*e1*, *e3*}, then {*e1*} $\bigcup$ {*e3*} = *se* where {*e1*} is generated from task $t2_1$ and {*e3*} is generated from task $t2_2$.
- $H^A(a)$ = the history of actual authorizations (i.e., a set of tuples) granted and revoked with the partial orders for an agent a where N(t) = a, $t \in$ T and $a \in$ A. The history stores four types of authorization tuples:
  - granted(*t*, *pr*, *x*) and revoked(*t*, *pr*, *x*) means the task t with privilege *pr* is granted/revoked to/from an agent a at time *x*.
  - granted(*d*, *pr*, *x*) and revoked(*d*, *pr*, *x*) means the document d with privilege *pr* is granted/revoked to/from an agent a at time *x*.

where (d, pr) $\in$ F $\circ$ G(t) and $x \in$ S(t). The partial order is determined by the time specified in the tuple. The workflow modeler can also grasp the order of document access for each task in order to build the relevant TAC. For illustration, here is an example to demonstrate information gathering using the partial order: $H^A(a)$ = { granted(*t1*, *execute*, *1*), granted(*d1*, *read*, *2*), granted(*d2*, *write*, *3*), revoked(*d1*, *read*, *4*), revoked(*d2*, *write*, *5*), revoked(*t1*, *execute*, *6*) }. From this history, we can deduce that the document *d1* with the privilege *read* is accessed before the document *d2* with the privilege *write* in the execution of task *t1*.

- S(t) = the session (time interval) for task t's execution where $t \in$ T. It has three cases:
  - *The task is started and completed* $S^{[]}$*:* If granted(t, "execute", x) $\in H^A(a)$ and revoked(t, "execute", y) $\in H^A(a)$ and y > x, then it is denoted as S(t) = [x, y]. The task is not active and the session is over, i.e., the session is the period from time x until time y. In this case, we define that $z \in$ S(t) if and only if $x \leq z \leq y$.
  - *The task is started and not completed* $S^{[)}$*:* If granted(t, "execute", x) $\in H^A(a)$ and revoked(t, "execute", y) $\notin H^A(a)$ and y > x, then it is denoted as S(t) = [x, $-$). The session is an open ended period from time x, i.e., the task is still active and the session is not yet over. In this case, we define that z $\in$ S(t) if and only if $x \leq z$.
  - *The task is not started and not completed* $S^{\emptyset}$*:* If granted(t, "execute", x) $\notin H^A(a)$, then S(t) = $\emptyset$. There is no session. In this case, we define that $\forall$ $z \in$ LT and $z \notin$ S(t).

  where "$-$" is null, N(t) = a, $a \in$ A and x, $y \in$ LT. Further, we define two functions *Begin(S(t))* and *End(S(t))* to assign the *begin* and *end* time for the session S(t). In case of $S^{[]}(t)$, then *Begin(S(t))* $\in$ LT and *End(S(t))* $\in$ LT. In case of $S^{[)}(t)$, then *Begin(S(t))* $\in$ LT and *End(S(t))* = "$-$". In case of $S^{\emptyset}(t)$, then there is no *Begin(S(t))* and *End(S(t))*.

- $H^T(t)$ = the history of actual events (i.e., a set of tuples) generated with the partial orders from a task t where $t \in$ T. It stores the event tuple: generated(e, x) meaning that event e is generated at time x where $e \in$ E and $x \in$ S(t).

A major objective of using histories H^A and H^T is to determine whether the workflow is secure or not, based on the state of the system. The state of the system at any one time is expressed as a set of values for all the state variables: {T, E, A, TAC, D, PR, I^T, I^E, I^A, I^{TAC}, I^D, C, C^{-1}, N, IE, OE, DT, P, F, G, H^A, H^T, S}. The definition of a secure state is a mathematical translation of the security requirements for a workflow into a set of invariants. An invariant is unchanged by specified mathematical operations or transformations. In a secure workflow model, there are three layers for a secure state: *Workflow, Data and Control*. Here we impose two security requirements to ensure the security properties of integrity, authorization and availability in the workflow layer and we also impose one security requirement to ensure the security properties of integrity and authorization in the data layer. The workflow modeler can impose other ad hoc security requirements in these three layers for different workflows. Here we only propose three fundamental security requirements for a generic workflow.

3.1 Workflow Layer

A workflow consists of a set of tasks that is interconnected by events. These tasks are executed by a set of agents. In our model, a task is an atomic workflow and each task is executed by exactly one agent N(t) = a where $t \in$ T and $a \in$ A. A secure workflow model needs to grant the agent the authorization(s) to execute a task(s) based on the occurrence of a certain event(s). The workflow layer involves the security properties of tasks, events and agents.

The security requirement to ensure the security property of availability in the workflow layer is: *"For every task there must be at least one agent who is able to execute the task."* The definition of the secure state is a mathematical translation of the security requirement into an invariant:

Invariant 1: The workflow can be executed if and only if, $\forall$ $t \in$ T,

- $C^{-1}(t) \neq \emptyset$.

Proof: The proof consists of two parts, (i) showing *Invariant 1* is necessary, and (ii) showing *Invariant 1* is sufficient.

From the workflow definition, we know that every workflow is represented as a partially ordered set of tasks that is coordinated by a set of events. The order of task execution is orchestrated by matching the input and output event(s) of each task. Therefore, it is obvious that there may be many legitimate paths for a workflow. Let one of the legitimate paths of a workflow be $[t_1, t_2, \ldots t_k]$. If there is a task t_i where i = 1 ... k that has no agent capable to execute it, i.e., $C^{-1}(t_i) = \emptyset$, this means that the subsequent task(s) $[t_{i+1}, \ldots, t_k]$ cannot be executed because each one of them is relying on the events(s) generated from the prior tasks, i.e., $t_i \in DT(t_{i+1})$ Thus, the workflow cannot be completed successfully.

To show that *Invariant 1* is sufficient we know that each task represents a piece of work that needs to be done by only one agent. Even though a task may have more than one agent capable to execute it, i.e., $|C^{-1}(t_i)| \geq 1$, only one of those agents will be assigned to execute the task, i.e., $N(t_i) \in C^{-1}(t_i)$. Thus, every task of the workflow can be executed.

A secure workflow model grants the privilege for an agent to execute the task if all its input events have been accessed and all its dependent tasks are completed. Further, it requires that all the authorizations for the agents that executed the dependent tasks must be revoked.

The security requirement to ensure the security properties of integrity and authorization in the workflow layer is: *"An agent can only execute the assigned task if and only if the privilege "execute" is granted. The secure workflow has to revoke the privilege from an agent if the task has completed execution."* If the "execute" privilege is revoked for task t from agent a at time x, then it means that the set of output events have been generated during the execution of task t (i.e., the events are generated before time x). If the "execute" privilege is granted for task t to agent a at time x, then it means that the set of dependent tasks have been revoked "execute" from the set of relevant agents (i.e., who executed the dependent tasks) before time x. The definition of the secure state is a mathematical translation of the security requirement into an invariant:

Invariant 2: An agent a can execute the assigned task t if and only if, $\forall\ t \in \mathrm{T}$, $\mathrm{N}(t) = a$ and $\exists\ x, y, z \in \mathrm{LT}$,

- if revoked(t, "execute", x) $\in H^A(a)$, then ($x \in \mathrm{S}(t) \bigwedge \{\text{generated}(e, y) \mid e \in se \text{ where } se \in \mathrm{OE}(t) \bigwedge x > y \bigwedge y \in \mathrm{S}(t)\} \subseteq H^T(t)$).
- if granted(t, "execute", x) $\in H^A(a)$, then $\forall_{t2 \in DT(t) and N(t2)=a2}(\text{revoked}(t2, \text{"execute"}, y) \in H^A(a2) \bigwedge x > y)$.

Proof: The proof consists of two parts, (i) showing *Invariant 2* is necessary, and (ii) showing *Invariant 2* is sufficient.

From the workflow definition, we know that a task t is completed once an output event set se is generated, i.e., $se \in \mathrm{OE}(t)$. All the events generated within the session of the task t are stored in a history, i.e., generated(e, y) $\in H^T(t)$ and $y \in \mathrm{S}(t)$. Thus, it is obvious that the output event set se must be the subset of the events stored in the history of the task t if and only if the output event set se is generated in the task t, i.e., $\{\text{generated}(e, y) \mid e \in se \text{ where } se \in \mathrm{OE}(t) \bigwedge y \in \mathrm{S}(t)\} \subseteq H^T(t)$. Referring to *Invariant 1*, for every task t there is an agent a to execute it, i.e., $\mathrm{N}(t) = a$. The authorization to execute the task t will be revoked from the agent a once the task t is completed, i.e., revoked(t, "execute", x) $\in H^A(a)$.

From the workflow definition, the order of task execution is orchestrated by matching the input and output event(s) of each task. Every task t (except the first task of a workflow) has a set of dependent tasks $\mathrm{DT}(t)$. There exists a combination of output events from the set of dependent tasks t_i that satisfy the input events of the task t, i.e., $t_i \in \mathrm{DT}(t)$. Referring to *Invariant 1*, for every task $t_i \in \mathrm{DT}(t)$ there is an agent a_i to execute it, i.e., $\mathrm{N}(t_i) = a_i$. Similarly, the authorization to execute t_i will be revoked from the agent a_i once the task t_i is completed. Once the authorization is revoked, there is no longer any event generated from the task t_i. If any task $t_i \in \mathrm{DT}(t)$ has not been completed, the task t cannot be initiated and the agent a will not be granted the authorization to execute the task t. The agent a will only be granted the authorization to execute the task t if and only if $\forall\ t_i \in \mathrm{DT}(t)$ have been completed, i.e., granted(t, "execute", x) $\in H^A(a)$.

To show that *Invariant 2* is sufficient we know that the task t is completed by the agent a if and only if the granted(t, "execute", x) $\in H^A(a)$ and revoked(t, "execute", y) $\in H^A(a)$ where $y > x$ because each task is only executed by one agent in our model, i.e., $\mathrm{N}(t) = a$. Thus, the agent a can only generate the event(s) from the task t (i.e., generated(e, z) $\in H^T(t)$ where $z \in \mathrm{S}(t)$) within the period from time x to time y (i.e., $x \leq z \leq y$).

3.2 Control Layer

The flow of authorizations is driven by the events generated from the task, and the control layer is the core engine of a secure workflow model. It involves the security properties of events and tasks. During the execution of a task, the agent generates different kinds of events such as the document access events. For example, "acquire(d, pr)", "release(d, pr)" or "process(d, pr)" where $d \in \mathrm{D}$ and $pr \in \mathrm{PR}$. These document access events will trigger the data layer to be executed. Note that the event(s) generated from the control layer are stored in the history of the task, i.e., $H^T(t)$.

3.3 Data Layer

In our model, we assume that all the data are stored as documents. Even though we develop this model by considering documents, it is applicable for any other kind of data or resources. During the execution of a task, an agent may need to access certain documents with different privileges. Each task has its specific order of document access flow that is described by a TAC. We model a set of events to acquire documents in the TAC. A secure workflow model needs to grant and revoke the document access privileges to an agent during the task execution. The data layer involves the security properties of tasks, agents and documents.

The security requirement to ensure the security properties of integrity and authorization in the data layer is: *"An agent can only access a document with a specific privilege if and only if the document access privilege is granted to the agent and also it is needed to access the document with the privilege during the task execution. The secure workflow has to revoke the document access privilege from an agent if the document access privilege is no longer needed."* The definition of the secure state is a mathematical translation of the security requirement into an invariant:

Invariant 3: A document access privilege is granted/revoked to/from an agent for a task if and only if, $\forall\ t \in \mathrm{T}$, $\mathrm{N}(t) = a$ and $\exists\ d \in \mathrm{D}$, $pr \in \mathrm{PR}$ and $x, y \in \mathrm{LT}$,

- if granted(d, pr, x) $\in H^A(a)$, then (d, pr) $\in \mathrm{F} \circ \mathrm{G}(t) \bigwedge x \in \mathrm{S}(t)$.
- if revoked(d, pr, y) $\in H^A(a)$, then granted(d, pr, x) $\in H^A(a) \bigwedge y > x \bigwedge y \in \mathrm{S}(t) \bigwedge x \in \mathrm{S}(t)$.

Proof: The proof consists of two parts, (i) showing *Invariant 3* is necessary, and (ii) showing *Invariant 3* is sufficient.

From the workflow definition, we know that the agent a can only generate event(s) for task t within the session of the task t. Based on our model, the agent a has to generate a document access event(s) to acquire documents. Therefore, the authorization of document access can only be granted to an agent (i.e., granted(d, pr, x) $\in H^A(a)$) within the session of a task (i.e., $x \in \mathrm{S}(t)$). There are three possible cases to grant an authorization:

- If granted(d, pr, x) $\notin H^A(a)$, then (d, pr) $\in \mathrm{F} \circ \mathrm{G}(t) \bigwedge x \in \mathrm{S}(t)$.
- If granted(d, pr, x) $\notin H^A(a)$, then (d, pr) $\notin \mathrm{F} \circ \mathrm{G}(t) \bigwedge x \in \mathrm{S}(t)$.
- If granted(d, pr, x) $\in H^A(a)$, then (d, pr) $\in \mathrm{F} \circ \mathrm{G}(t) \bigwedge x \in \mathrm{S}(t)$.

From the workflow definition, the agent a can only have the authorizations to access the documents with specific privileges specified in $\mathrm{F} \circ \mathrm{G}(t)$ during the session of task $\mathrm{S}(t)$. Therefore, there is no such case like:

- If granted(d, pr, x) $\in H^A(a)$, then (d, pr) $\notin \mathrm{F} \circ \mathrm{G}(t) \bigwedge x \in \mathrm{S}(t)$.

Similarly, there are three possible cases to revoke an authorization:

- if revoked(d, pr, y) $\notin H^A(a)$, then granted(d, pr, x) $\in H^A(a) \bigwedge (y > x) \bigwedge (y \in \mathrm{S}(t)) \bigwedge (x \in \mathrm{S}(t))$.
- if revoked(d, pr, y) $\notin H^A(a)$, then granted(d, pr, x) $\notin H^A(a) \bigwedge (y > x) \bigwedge (y \in \mathrm{S}(t)) \bigwedge (x \in \mathrm{S}(t))$.

- if revoked(d, pr, y) $\in H^A(a)$, then granted(d, pr, x) $\in H^A(a) \bigwedge (y > x) \bigwedge (y \in \mathrm{S}(t)) \bigwedge (x \in \mathrm{S}(t))$.

It is infeasible to revoke a non-existent authorization from an agent. Therefore, there is no such case like:

- if revoked(d, pr, y) $\in H^A(a)$, then granted(d, pr, x) $\notin H^A(a) \bigwedge (y > x) \bigwedge (y \in \mathrm{S}(t)) \bigwedge (x \in \mathrm{S}(t))$.

To show that *Invariant 3* is sufficient. Referring to the three possible cases for an authorization granted, we only consider the authorization really granted to an agent, i.e., granted(d, pr, x) $\in H^A(a)$. Similarly, we only consider the authorization really revoked from an agent, i.e., revoked(d, pr, y) $\in H^A(a)$. If the time x and y are within the session of task t (i.e., $x \in \mathrm{S}(t)$ and $y \in \mathrm{S}(t)$), the authorization(s) is/are granted and revoked to/from the agent a during the execution of task t.

3.4 Authorization Functions

An authorization function can be viewed as a procedure call (transition function) to a service routine, where the service desired is a specific change to the security relevant state variables. The parameters to the function are specified and must be validated before the model carries out any state change. The authorization functions in a secure workflow model are used to assign a task to an agent, grant and revoke certain privileges of tasks or documents to an agent, and generate events during the task execution. There are five major authorization functions in a secure workflow model. The prime symbolic variables are a new state of variables after certain functions have executed. The authorization function must reflect the properties of those invariants imposed in the secure workflow model. In this model, we also use a function *timestamp()* that gives the current discrete time as a positive integer Z^+.

- To ensure the property of authorization, the secure workflow model assigns the task to an agent if and only if the agent can execute the task.

Function 1: Assign(t,a)
if t ∈ T and a ∈ A {
  if t ∈ C(a) and a ∈ C^{-1}(t) {
    N(t)′ = a
  }
}

- To ensure the properties of integrity and authorization, the secure workflow model grants the task to the assigned agent for execution if and only if the set of input events is generated, the task is not started and all the dependent tasks are completed in the relevant session.

Function 2: GrantT(t,a,se)
x ∈ LT;
if t ∈ T and a ∈ A and se ∈ IE(t) {
  if N(t) == a and S(t) == ∅ {
    for all t_{DT} ∈ DT(t) {
      if revoked(t_{DT}, "execute", x) ∉
      H^A(N(t_{DT})) and x ∈ S(t_{DT}) {
        exit
      }
    };
    H^A(a)′ = H^A(a) ⋃ {granted(t, "execute", timestamp())};
    Begin(S(t)′) = timestamp()
  }
}

- To ensure the properties of integrity and authorization, the secure workflow model revokes the task from the assigned agent if and only if the set of output events is generated and all the granted privileges for documents are revoked in the session.

Function 3: RevokeT(t,a,se)
d ∈ D;
pr ∈ PR;
x, y ∈ LT;
if t ∈ T and a ∈ A and se ∈ OE(t) {
  if N(t) == a and timestamp() ∈ S(t) {
    for all granted(d, pr, x) ∈ H^A(a) and
    revoked(d, pr, y) ∉ H^A(a) and
    y > x and x ∈ S(t) and y ∈ S(t) {
      H^A(a)′ = H^A(a) ⋃ {revoked(d, pr, timestamp())}
    };
    H^A(a)′ = H^A(a) ⋃ {revoked(t, "execute", timestamp())};
    End(S(t)′) = timestamp()
  }
}

- To ensure the properties of integrity and authorization, an agent can generate the event for a task if and only if it is authorized in the session.

Function 4: GenerateE(t,a,e)
if t ∈ T and a ∈ A and e ∈ E {
  if N(t) == a and timestamp() ∈ S(t) {
    H^T(t)′ = H^T(t) ⋃ {generated(e, timestamp())}
  }
}

- To ensure the properties of integrity and authorization, the secure workflow model grants the document access privilege to the agent for execution if and only if it is authorized by the task's TAC in the session.

Function 5: GrantD(t,a,e)
d ∈ D;
pr ∈ PR;
x, y ∈ LT;
if t ∈ T and a ∈ A and e ∈ E {
  if N(t) == a and e == acquire(d, pr) and
  (d, pr) ∈ F ∘ G(t) {
    if not ∃ (granted(d, pr, x) ∈ H^A(a) and
    revoked(d, pr, y) ∉ H^A(a) and
    y > x and x ∈ S(t) and y ∈ S(t)) and
    timestamp() ∈ S(t) {
      H^A(a)′ = H^A(a) ⋃ {granted(d, pr, timestamp())}
    }
  }
}

- To ensure the properties of integrity and authorization, the secure workflow model revokes the document access privilege from the agent if and only if the document access privilege or task is completed in the session.

Function 6: RevokeD(t,a,e)
d ∈ D;
pr ∈ PR;
x, y ∈ LT;
if t ∈ T and a ∈ A and e ∈ E {
  if N(t) == a and e == release(d, pr) and
  timestamp() ∈ S(t) {
    if granted(d, pr, x) ∈ H^A(a) and
    revoked(d, pr, y) ∉ H^A(a)
    and y > x and x ∈ S(t) and y ∈ S(t) {
      $H^A(a)' = H^A(a) \bigcup$
      {revoked(d, pr, timestamp())}
    }
  }
}

3.5 Multi-Layered State Machine for Secure Workflow Model

Figure 1 shows a multi-layered state machine for a secure workflow model based on the authorization functions as discussed in the previous section. This multi-layered state machine can serve as a powerful tool for modeling the secure workflow model at a conceptual and logical level from the aspects of workflow, control and data. The authorization functions involved (denoted as "*") in the state machine at different layers: workflow, control and data are:

Authorization Function	Workflow Layer	Control Layer	Data Layer
Assign(t,a)	*		
GrantT(t,a,se)	*		
RevokeT(t,a,se)	*		
GenerateE(t,a,e)		*	
GrantD(t,a,e)			*
RevokeD(t,a,e)			*

Further, the state variables referenced (denote as "*") by the authorization functions and invariants in the state machine at different layers are:

State Variable	Workflow Layer	Control Layer	Data Layer
T	*	*	*
E	*	*	*
A	*	*	*
TAC			*
D			*
PR			*
I^T	*		
I^E		*	
I^A	*		
I^{TAC}			*
I^D			*
C	*		
C^{-1}	*		
N	*	*	*
IE	*	*	
OE	*	*	
DT	*	*	
P			*
F			*
G			*
H^A	*		*
H^T	*	*	
S	*	*	*

4 Conclusions

In this paper, we develop a secure workflow model using a multi-layered state machine to manage and monitor the flow of authorizations at different layers for a secure workflow execution. A state-machine model describes a system as an abstract mathematical state machine with a set of transition functions. A multi-layered state machine describes a system in different layers where each layer is an abstract mathematical state machine with a set of transition functions. The interaction between two mathematical state machines at different layers is triggered by an event. The novel part of our secure workflow model is separating the various aspects of control in a workflow and portraying it as a multi-layered architecture for analyzing the flow of authorizations. There are three layers in a secure workflow: *workflow*, *control* and *data*. The secure workflow model is used to ensure the security properties of integrity, authorization, and availability. We transform the security requirements into a set of invariants for different layers in the secure workflow model. Further, we describe a set of authorization functions to support the state machine.

References

Atluri, V. & Huang, W.-K. (1996), An Authorization Model for Workflows, *in* 'Proceedings of the Forth European Symposium on Research in Computer Security', pp. 44-64.

Atluri, V. & Huang, W.-K. (1996), An Extended Petri Net Model for Supporting Workflows in a Multilevel Secure Environment, *in* 'Proceedings of the 10th IFIP WG 11.3 Working conference on Database Security', pp. 240-258.

Atluri, V., Huang, W.-K. & Bertino, E. (1997), An Execution Model for Multilevel Secure Workflows, *in* 'Proceedings of the 11th IFIP Working Conference on Database Security', pp. 151-165.

Booch, G., Rumbaugh, J. & Jacobson, I. (1999), *The Unified Modeling Language User Guide*, Addison-Wesley.

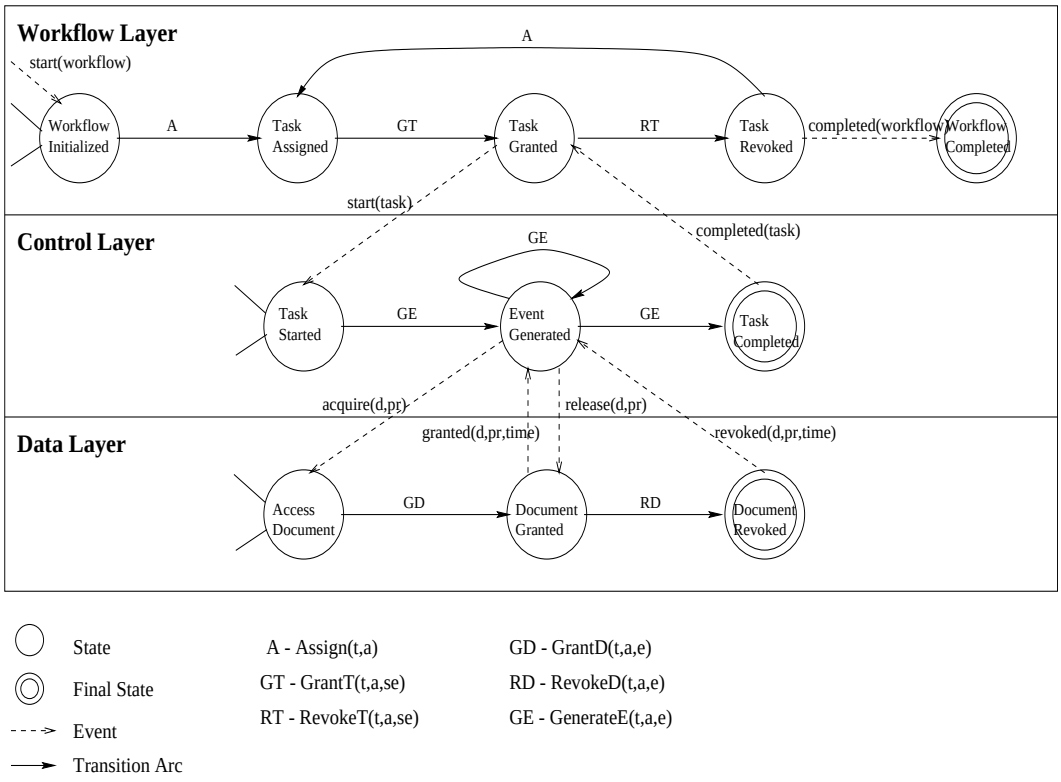

Figure 1: Multi-Layered State Machine for Secure Workflow Model

Hollingsworth, D. (1995), Workflow Management Coalition (WfMC), The Workflow Reference Model, Document Number TC00-1003, Document Status - Issue 1.1.

Huang, W.-K. & Atluri, V. (1999), SecureFlow: A Secure Web-enabled Workflow Management System, *in* 'Proceedings of the 4th ACM Workshop on Role-Based Access Control', pp. 83-94.

Hung, P. C. K. (2001), Secure Workflow Model, Ph.d., Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong.

Hung, P. C. K. (2002), 'Specifying Conflict of Interest in Web Services Endpoint Language (WSEL)', *The ACM SIGecom Exchanges* **3**(3), 1–8.

Joshi, J. B. D., Aref, W. G., Ghafoor, A. & Spafford, E. H. (2001), 'Security Models for Web-based Applications', *Communications of the ACM* **44**(2), 38–44.

Kang, M. H., Park, J. S. & Froscher, J. N. (2001), Access Control Mechanisms for Inter-organizational Workflow, *in* 'Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies', pp. 66-74.

Michelis, G. D. (1999), Net Theory and Workflow Models, *in* 'Proceedings of the 20th International Conference in Application and Theory of Petri Nets', pp. viii+423.

Nyanchama, M. & Osborn, S. (1999), 'The Role Graph Model and Conflict of Interest', *ACM Transactions on Information and System Security* **2**(1), 3–33.

Pernul, G. (1992), Security Constraint Processing During Multilevel Secure Database Design, *in* 'Proceedings of Eighth Annual IEEE Computer Security Applications Conference', pp. 229-247.

Thuraisingham, B., Clifton, C., Gupta, A., Bertino, E. & Ferrari, E. (2001), Directions for Web and E-commerce Applications Security, *in* 'Proceedings of Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises', pp. 200-204.

WfMC (2001), Workflow Management Coalition (WfMC), Workflow Security Considerations - White Paper, Document Number WFMC-TC-1019, Document Status - Issue 1.0.