An Authorization Model for Workflows*

Vijayalakshmi Atluri and Wei-Kuang Huang

Center for Information Management, Integration, and Connectivity (CIMIC)
and
MS/CIS Department
Rutgers University
180 University Avenue, Newark, NJ 07102
{atluri,waynexh@andromeda.rutgers.edu}

Abstract. Workflows represent processes in manufacturing and office environments that typically consist of several well-defined activities (known as tasks). To ensure that these tasks are executed by authorized users or processes (subjects), proper authorization mechanisms must be in place. Moreover, to make sure that authorized subjects gain access on the required objects only during the execution of the specific task, granting and revoking of privileges need to be synchronized with the progression of the workflow. A predefined specification of the privileges often allows access for more than the time required, thus, though a subject completes the task or have not yet begun the task, it may still possess privileges to access the objects, resulting in compromising security.
In this paper, we propose a *Workflow Authorization Model* (WAM) that is capable of specifying authorizations in such a way that subjects gain access to required objects only during the execution of the task, thus synchronizing the *authorization flow* with the workflow. To achieve this synchronization, we associate an *Authorization Template* (AT) with each task, which allows appropriate authorizations to be granted only when the task starts and to revoke them when the task finishes. In this paper, we also present a model of implementation based on *Petri nets* and show how this synchronization can be implemented. Because the theoretical aspects of Petri nets have been extensively studied and due to their strong mathematical foundation, a Petri net representation of an authorization model serves as a good tool for conducting safety analysis since the safety problem in the authorization model is equivalent to the reachability problem in Petri nets.
Key Words: Security, Authorization, Workflow, Petri nets

1 Introduction

Workflows typically represent processes involved in manufacturing and office environments and heterogeneous database management systems. The various activities in a workflow can usually be separated into well defined tasks. These tasks in turn are related and dependent on one another, and therefore need to

* This work was supported in part by the National Science Foundation grant IRI-9624222.

be executed in a coordinated manner. Execution of these separate tasks can either be carried out by humans, processes such as an application program, or a database management system.

To ensure that these tasks are executed by authorized subjects (users or processes), appropriate authorization mechanisms must be in place. A suitable authorization model for workflows must ensure that authorization is granted only when the task starts and revoked as soon as the task finishes. Otherwise, a subject may possess authorization for time periods longer than required, which may compromise security.

For example, consider a workflow that represents document release process in an organization. Assume this workflow consists of the following two tasks: A scientist prepares a document (task 1) and then prior to releasing this document gets approval from the patent-officer of his organization (task 2). During execution of task 1, the scientist will have all the rights on the document (read, write etc.). When he submits it to the patent-officer, a read authorization on the document has to be granted to the patent-officer. Since the scientist should not be allowed to modify the document during or after the approval process, the write privilege of the scientist on the document must be revoked as soon as he submits it for approval. This is required to ensure that the scientist is not able to alter the document during or after the approval. (Such non-monotonic authorization models can be found, for example, in [22].) In the paper world, since the patient-officer receives a hard copy of the document, and therefore it is no longer with the scientist, the revocation of the write privilege from the scientist will be automatically accomplished. In case of electronic documents, a proper authorization model should mimic such a scenario. In order to ensure authorized subjects are granted privileges on required objects only while the task is being executed, propagation of authorization (i.e. *authorization flow*) has to be synchronized with the workflow.

To our knowledge, no authorization model can be found in the literature that addresses this issue of achieving synchronization between authorization flow and workflow. Recently, several extensions to the basic authorization model can be seen in number of directions. One direction deals with increasing the expressive power of the authorization models and developing appropriate tools and mechanisms to support these models. These include introducing negative authorization [3, 17], role-based and task-based authorizations and separation of duties [6, 19, 20], and temporal authorization [2]. Other direction deals with extending authorization models for advanced DBMSs such as object-oriented [8, 17, 4] and distributed [25, 18, 11] DBMSs.

However, in these models, authorizations are in general specified with respect to object, right, subject granting right, subject receiving right, etc. These models study the propagation of authorization but do not tie it with any activity in the system. However, as seen from the above example, authorization flow need to be tightly coupled to the workflow in order to ensure subjects possess authorizations only when required. [21] also recognizes that more sophisticated approaches than the conventional access control techniques are required when dealing with situations that control operation sequences.

Although models exist that allow specification of authorizations associated with a time interval, they are not suitable for workflow environments. For example, recently, an authorization model to represent temporal privileges has been proposed by Bertino et al. [2]. It allows authorizations to be specified on objects for specified time intervals. In this model a temporal authorization is specified as `(time, auth)`, where `time` = $[t_b, t_e]$ is a time interval, and `auth` = `(s,o,m,pn,g)` is an authorization. Here t_b and t_e represent the beginning and ending time during which `auth` is valid, `s` represents the subject, `o` the object, `m` the privilege, `pn` whether negative or positive authorization, and `g` the grantor of the authorization.

Since in this model authorization is granted during a predefined and fixed time interval, it is not suitable when dealing with workflows because appropriate authorizations should be granted or revoked synchronously with the starting and ending of a task. This is because it is difficult to predict the actual execution time of each task in many workflow situations and therefore not possible to determine their time interval in advance.

For example, imagine the following scenario: Suppose the authorization for relevant subjects on required object to execute a task has been specified with time interval $[t_b, t_e]$. Also assume the task actually starts at t_s and finishes by t_f. Consider the following three cases: (1) if $[t_b, t_e]$ is same as $[t_s, t_f]$, then authorization is valid exactly during the execution of the task (2) if $[t_s, t_f]$ is within $[t_b, t_e]$ then authorization is valid for a longer duration than required (3) if $[t_b, t_e]$ is within $[t_f, t_s]$ or $[t_b, t_e]$ overlaps $[t_f, t_s]$, then either the authorization is not valid when needed or valid for a longer period than required.

Even if one can determine the temporal interval during which a task has to be executed, delays experienced by one task may propagate to subsequent tasks thereby delaying their execution. Thus by the time the task actually starts, the required authorization may expire. On the other hand, it is not practical for a security administrator to monitor the workflow and grant (or revoke) authorizations accordingly. Although no formal authorization models exist that can synchronize authorization flow with the workflow, in some commercial Workflow Management Systems (WFMSs) such as Lotus Notes, this can be simulated by embedding scripts that test for the completion of the task and thereby revoke authorizations for individuals who have performed the task.

Bertino et al.'s model also allows operations such as `WHENEVER`, `ASLONGAS`, `WHENEVERNOT` and `UNLESS` to be specified on authorizations, where `WHENEVER` states that a subject s_i can gain a privilege on object o `WHENEVER` subject s_j has the same privilege on o. (`ASLONGAS`, `WHENEVERNOT` and `UNLESS` can also be interpreted similarly.) Since specification of authorizations is not based on the tasks this model is not completely suitable for workflow environments.

In addition to synchronizing the authorization flow with the workflow, there are several other desirable features that a suitable workflow authorization model must possess. We take a concrete example to illustrate these features.

Example 1. Consider a workflow that represents the selection process of research papers for a conference. This workflow consists of a number of tasks including collection of papers from the authors, distribution of papers to selected reviewers, generation of the reviews, summarizing all the reviews, forwarding the summary and decision of the conference chair to the authors and then finally announcing the list of selected papers to the research community. Assume that three individuals are required to review each paper. Authors are given privileges to create objects thereby anybody can submit a paper to the conference by creating an object. Authors will have read and write privileges on their paper although the write privilege is associated with a time limit since it has to be revoked after the deadline for submission of the papers. The conference chair then selects the reviewers. Though anyone among the set of reviewers may play the role of a reviewer, for any given paper, the three reviewers have to be different individuals and they must not be the authors of the paper. Reviewers send their responses by creating an object on which only read privilege has to be given to the conference chair. The conference chair then produces a summary of all the three reviews which is accessible only to the chair and the authors of the paper at which point authors are given back the write privileges on the paper. The final decision as to whether the paper has been accepted or rejected is produced and the list of accepted papers will be created by the conference chair, and this object is public and anyone can gain read privileges on this object. This workflow has been depicted in figure 1.

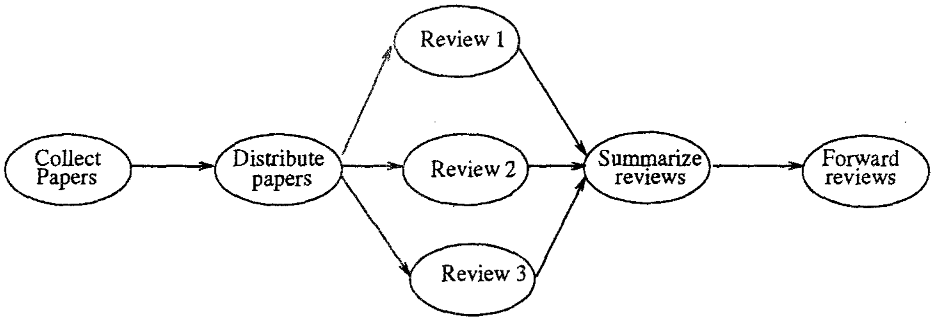


Fig. 1. Workflow Representing the Paper Reviewing Process

As seen from the example above, a workflow authorization model should support the following features. We explain below why these are required with the above example.

1. Authorizations have to be granted to subjects only during the execution of the task and must be revoked immediately after the completion of the task. This is required to guarantee that an author cannot modify the paper after submitting it to the conference chair.

2. Capability to handle temporal constraints is required to express conditions such as a reviewer must return the reviews by May 1996.
3. A role-based authorization is required to express an authorization such as "anyone in the program committee may act as a reviewer."
4. Separation of duties, which is necessary to prevent a single person providing all the three reviews on a paper, or an author of the paper reviewing his/her own paper.
5. Event-based authorization is required because the paper has to be made public and anyone can gain read access on the paper only if it is accepted for publication. Otherwise, the author(s) gain all the privileges on the paper. In other words, authorizations on objects and tasks may depend on the outcome of a task (such as its success or failure, or even the value of the outcome of the task).

In this paper, we propose a *Workflow Authorization Model* (WAM) that can *dynamically* change the time interval that specifies during which the authorization is valid. By the term dynamic, we mean that the time interval associated with the required authorization to perform a task changes according to the time during which the task actually executes. Our model addresses the first two issues enumerated above, i.e., synchronization of authorization flow with the workflow and specification of temporal constraints. In WAM, we introduce the notion of *Authorization Template* (AT) which specifies the static parameters of the authorization that can be defined during the design of the workflow and can be attached to the task. When the task actually starts execution, AT is used to derive the actual authorization. Section 2 presents WAM and shows how synchronization of authorization flow and workflow can be achieved.

We have presented a model of implementation of WAM using Petri nets, in which we show how synchronization of authorization flow and workflow can be achieved. Petri nets have been widely used for modeling synchronous activities. Their graphical nature and ability to model concurrent processes in a natural manner make them excellent tools for modeling workflow authorization. Moreover, because of their strong mathematical foundation, Petri nets are good tools for analysis of the authorization models. The main reason for adapting Petri nets is that *the safety problem*[2] *in authorization models is equivalent to the reachability problem in Petri nets.* Since Petri nets have been extensively studied with respect to this issue, representation of WAM allows us to adapt the existing reachability analysis techniques of Petri nets to conducting the safety analysis of WAM. (Similar representations such as finite state automata for the enforcement of security specifications [5] do not possess the aforementioned advantages of Petri nets.) We use a combination of two extended Petri net models – colored and timed Petri nets – to represent WAM as a Petri net. The Petri net representation of WAM is presented in section 3.

In section 4, we have presented a brief discussion of how WAM can be extended to specify role-based authorizations and separation of duties.

[2] The safety problem can be stated as the following question: "Is there a reachable state in which a particular subject possesses a particular privilege for a specific object?"

2 Workflow Authorization Model (WAM)

In this section, we propose a workflow authorization model (WAM) and show how required authorizations can be assigned to subjects so that the time interval during which an authorization is valid can be synchronized with workflow execution.

A workflow W can be represented as a *partially ordered* set of tasks $\{w_1, w_2, \ldots w_n\}$, where each task w_i in turn can be defined as a set OP_i of a partial or total order of operations $\{op_1, op_2 \ldots op_n\}$ that involve manipulation of objects [9].

Most workflows can be perceived as coordinated execution of tasks that involve processing of relevant objects by subjects (either humans or programs). Thus one can imagine every task starts when one or more objects arrive to the task for processing, and when the task finishes these objects leave. For example, consider the workflow of check processing consisting of three tasks: (1) preparation, (2) approval and (3) issue of the check. For the second task, i.e., check approval, to start, the object (check) has to be sent from the previous task (i.e., check preparation). When the approval task finishes, the approved check has to be sent to the following task, i.e., check issuing. Therefore, we assume a task starts only when one or more objects arrive to the task and when it finishes one or more objects leave the task. If there exists no such object moving from one task to the other, then one can assume the signal which acts as an input to trigger the starting of the task as an object.

Processing of a task involves accessing certain objects by certain subjects with certain privileges. To execute a task w_i, relevant privileges on required objects have to be granted to appropriate subjects. In this section, we develop several definitions to construct WAM.

Let $S = \{s_1, s_2 \ldots\}$ denote the set of subjects, $O = \{o_1, o_2 \ldots\}$ the set of objects $\Gamma = \{\gamma_1, \gamma_2 \ldots\}$ a finite set of objects types. The function $Y : O \rightarrow \Gamma$. That is, if $Y(o_i) = \gamma_j$, then o_i is of type γ_j. Let PR denote a finite set of privileges.

The following defines a time set and a time interval.

Definition 1.

1. Time set $\mathcal{T} = \{\tau \in \mathcal{R}^3 \mid \tau \geq 0\}$, and
2. a time interval $\{[\tau_l, \tau_u] \in \mathcal{T} \times \mathcal{T} \mid \tau_l \leq \tau_u\}$ represents the set of all closed intervals. □

The above definition dictates that the interval is guided by an lower and a upper boundary, τ_l and τ_u, respectively. We say an interval $[\tau_1, \tau_2]$ is within $[\tau_3, \tau_4]$ iff $\tau_3 \leq \tau_1$ and $\tau_4 \geq \tau_2$ and a time τ_1 is within $[\tau_3, \tau_4]$ iff $\tau_3 \leq \tau_1 \leq \tau_4$.

Definition 2. We define a *task* w_i as: $(OP_i, \Gamma_{IN_i}, \Gamma_{OUT_i}, [\tau_{l_i}, \tau_{u_i}])$, where OP_i is the set of operations to be performed in w_i, $\Gamma_{IN_i} \subseteq \Gamma$ is the set of object types

[3] $\mathcal{R}$ represents the set of all real numbers.

allowed as inputs, $\Gamma_{OUT_i} \subseteq \Gamma$ is the set of object types expected as outputs, and $[\tau_{l_i}, \tau_{u_i}]$ is the time interval during which w_i must be executed. □

Here $[\tau_{l_i}, \tau_{u_i}]$ specify temporal constraints stating the lower and upper bounds of the time interval during which a task is allowed to be executed. As an example, a temporal constraint may be specified as follows: "a task for preparing a check must be started after time = 10 and finished by time = 20."

Definition 3. We define a *task-instance*, *w-inst*$_i$ as: $(OPER_i, IN_i, OUT_i, [\tau_{s_i}, \tau_{f_i}])$ where $OPER_i$ is the set of operations performed during the execution of w_i, IN_i is the set of input objects to w_i such that $IN_i = \{x \in O | Y(x) \in \Gamma_{IN_i}\}$, OUT_i is the set of output objects from w_i such that $OUT_i = \{x \in O | \gamma(x) \in \Gamma_{OUT_i}\}$, and $[\tau_{s_i}, \tau_{f_i}]$ is the time interval during which w_i has been executed. □

Whenever a task is executed, a task-instance will be generated. Thus, a task w_i may generate several *w-inst*$_i$'s. τ_{s_i} and τ_{f_i} in the above definition indicate the time at which that particular task-instance has started and finished execution, respectively, whereas $[\tau_{l_i}, \tau_{u_i}]$ represent the time during which the task must be executed. Note that $[\tau_{l_i}, \tau_{u_i}]$ may differ from $[\tau_{s_i}, \tau_{f_i}]$, however, to ensure the temporal constraints, $[\tau_{s_i}, \tau_{f_i}]$ must be within $[\tau_{l_i}, \tau_{u_i}]$. To guarantee the above requirement, we use a model based on Petri nets.

Definition 4. We define an authorization as a 4-tuple $A = (s, o, pr, [\tau_b, \tau_e])$, where subject s is granted access on object o with privilege pr at time τ_b and is revoked at time τ_e. □

Definition 5. Given a task w_i, we define an authorization template $AT(w_i)$ as a 3-tuple $AT(w_i) = (s_i, (\gamma_i, -), pr_i)$ where
(i) $s_i \in S$,
(ii) $(\gamma_i, -)$ is an *object hole* which can be filled by an object o_i of type γ_i, and
(iii) pr_i is the privilege to be granted to s_i on object o_i when $(\gamma_i, -)$ is filled by o_i. □

In the definition for $AT(w_i)$ (i) says that only subject s_i is allowed to execute task w_i, (ii) dictates that only objects of type γ_i can be processed by w_i thus the object hole $(\gamma_i, -)$ allows objects of only type γ_i to be filled in, and (iii) says that s_i requires a privilege pr_i on the objects that arrive at w_i for processing.

Authorization templates are attached to the tasks in a workflow.[4] A task w_i may have more than one authorization template attached to it. More ATs are required in cases where there are more than one type of object to be processed, or more than one subject is required to perform the processing.

[4] The notion of templates can also be found in systems such as Hydra [26] where a template is defined as (type, required-rights). This is used to generate a new capability for an object by checking the type and rights specified in the template with its type and existing capability. Our notion of authorization template is different from that is Hydra in the sense that it grants a new authorization to a subject on the specified object if the object's type is same as that specified in the template.

To distinguish the subjects and privileges in AT from those in A, we often use $s(AT)$ and $pr(AT)$.

An authorization template allows us to specify rules such as "A subject John is allowed to perform check preparation." These can actually be stated during the design process by the workflow designer. However, the authorization to prepare the check is granted to John only when the task of check preparation actually starts. And this privilege will be revoked when this task is completed. The following authorization derivation rule ensures this.

Definition 6 Authorization Derivation Rule. Given an authorization template $AT(w_i) = (s_i, (\gamma_i, -), pr_i)$ of task $w_i = (OP_i, \Gamma_{IN_i}, \Gamma_{OUT_i}, [\tau_{l_i}, \tau_{u_i}])$, an authorization $A_i = (s_i, o_i, pr_i, [\tau_{b_i}, \tau_{e_i}])$ is derived as follows:
Grant Rule: Suppose object $o_i \in \Gamma_{IN_i}$ is sent to w_i at τ_{a_i} to start w_i. Let the starting time of w_i be τ_{s_i}.
If $\tau_{a_i} \leq \tau_{u_i}$, then $s_i \leftarrow s(AT)$, $pr_i \leftarrow pr(AT)$, $\tau_{e_i} \leftarrow \tau_{u_i}$, and
(if $\tau_{a_i} \leq \tau_{l_i}$ then $\tau_{b_i} \leftarrow \tau_{l_i}$; otherwise $\tau_{b_i} \leftarrow \tau_{a_i}$)
Revoke Rule: Suppose w_i ends at τ_{f_i} at which point o_i leaves w_i.
If $\tau_{f_i} \leq \tau_{u_i}$, then $\tau_{e_i} \leftarrow \tau_{f_i}$. □

We explain below how authorizations are derived from the authorization templates. Suppose a workflow consists of two tasks w_i and w_j. Also suppose there exists a temporal constraint on w_i which states that w_i must be executed only during the time interval $[\tau_{l_i}, \tau_{u_i}]$. Assume executing w_i involves processing and therefore accessing object o which is of type γ_i. To start w_i, an object o of type γ_i is first sent as an input to w_i.[5] After completing the execution, w_i passes the object o to the next task w_j. The authorization templates associated with w_i and w_j are: $AT(w_i) = (s_i, (\gamma_i, -), pr_i)$ and $AT(w_j) = (s_j, (\gamma_j, -), pr_j)$ (in this case γ_i may equal γ_j).

Execution of w_i starts when o arrives to w_i. Let this time be τ_{a_i}. If τ_{a_i} is within the specified time interval $[\tau_{l_i}, \tau_{u_i}]$ then w_i will be started, and the object hole in the authorization template is filled with o. At this point, the corresponding authorization A_i is derived according to the authorization derivation rule as follows: If $\tau_{a_i} \geq \tau_{l_i}$, then the time at which the object arrives (τ_{a_i}) is assigned to τ_{b_i}, and the specified time before which the task must complete (τ_{u_i}) is assigned to τ_{e_i}. Assigning τ_{u_i} as τ_{e_i} is required to ensure that A_i is not valid after the specified interval even if the task is still executing beyond the upper bound specified in the time constraint of w_i. However, if $\tau_{a_i} < \tau_{l_i}$, then τ_{l_i} is assigned to both τ_{b_i}. That is, even if the object arrives earlier than the specified time interval, authorization is valid only from τ_{l_i} but not from the time at which the object arrives. If $\tau_{a_i} > \tau_{u_i}$ then the object is rejected. Thus s_i is given privilege pr_i on o only when w_i starts execution.

When w_i finishes its execution (say at τ_{f_i}), o is passed on to w_j. Now, the object hole in the authorization template $AT(w_j)$ is filled with o, while that in $AT(w_i)$ becomes empty. If $\tau_{f_i} < \tau_{u_i}$ then τ_{e_i} in A_i is modified such that $\tau_{e_i} =$

[5] If there are no input objects to be sent to w_i, one can imagine that the input to start a task can be treated as an object or a dummy object can be assumed.

τ_{f_i}. Otherwise, τ_{e_i} is not modified. Thus s_i has privilege pr_i on o only until τ_{f_i}, i.e., only during the execution of w_i and is taken away from it as soon as w_i is completed. Even if w_i does not complete by time τ_{u_i}, A_i is valid only until τ_{u_i}. The validity of authorization is therefore guided by the specified duration. The authorization thus created showing the duration that the authorization has been granted for a particular task can be used for auditing purposes. In the following, we explain the process of deriving authorizations by taking a real example.

Example 2. Consider once again the check processing workflow, which involves the following three tasks, w_1, w_2 and w_3 denoting prepare check, approve check and issue check, respectively. They can be expressed as follows:

w_1 = ({read request, prepare check}, {request, check}, {check}, [10,50])
w_2 = ({approve check}, { check}, { check}, [20,60])
w_3 = ({issue check}, { check}, { check}, [40,80])

Suppose the associated subjects for performing these processes are John, Mary, and Ken, respectively. Now, instead of granting all the required privileges for every involved staff in advance, we first create the following authorization templates. (Appropriate authorizations to perform these tasks are not enforced until the tasks are actually processed.)

$AT_1(w_1)$ = (John, (request,-), read)
$AT_2(w_1)$ = (John, (check,-), prepare)
$AT(w_2)$= (Mary, (check,-), approve)
$AT(w_3)$= (Ken, (check,-), issue)

Now suppose the requests for payment arrive as follows.

Request $rq1$ at 40
Request $rq2$ at 55.

Before any task starts, no one in the workflow has been granted any valid authorization. At 40, the object $rq1$ arrives to w_1. This object is filled into the authorization template $AT_1(w_1)$ = (John, (request,-), read), thereby generating an authorization (John,$rq1$, read, [40,50]). According to w_1, a new object check, say $ck023$ is created at w_1. Thus $AT_2(w_1)$ = (John, check(-), prepare) is filled with $ck023$ thus generating another authorization (John, $ck023$, prepare, [40,50]).

Suppose John finishes w_1 at 47, then the authorizations on $rq1$ and $ck023$ are revoked for John by replacing the upper bound with 47, thus forming the authorizations as (John, $rq1$, read, [40,47]), and (John, $ck023$, prepare, [40,47]). Also note that, at this point, a task-instance $w\text{-}inst_1$ = ({read $rq1$, prepare $ck023$}, {$rq1$}, {$ck023$}, [40,47]) will also be generated.

Suppose at 47, check_1 is sent to w_2. Then the authorization template $AT(w_2)$= (Mary, (check,-),approve) would be filled, thus generating the authorization (Mary, $ck023$, approve, [47, 60]). Assume w_2 completes at 54. Then the authorization is changed to (Mary, ($ck023$, approve, [47,54]). At this point the object $ck023$ is sent to w_3 which fills $AT(w_3)=$ (Ken, (check,-), issue) and generates an authorization (Ken, $ck023$, issue, [54,80]). After the completion of this task (say at 60) this authorization changes to (Ken, $ck023$, issue, [54,60]).

However, in case of $rq2$, since the upper limit in $w_1 =$ ({read request, prepare check}, {request}, {check}, [10,50]) is lower than the time at which $rq2$ arrives at w_1 (55) the authorization template does not generate an authorization. Thus the workflow cannot be started for $rq2$, and therefore, there will not be a task-instance for $rq2$.

For the sake of simplicity, indeed, in this example we have omitted details such as when a check is issued, it has to be assigned to an account and appropriate authorizations such as read, write (to perform debit) have to be granted on this account.

3 A Petri net representation of the Workflow Authorization Model

In this section, we present a model of implementation of WAM, which is based on Petri nets (PN). PNs are a graphical as well as a mathematical modeling tool. As a graphical tool PNs provide visualization (similar to flow charts, block diagrams, and the like) of the workflow process, and as a mathematical tool PNs enable analysis of the behavior of the workflow. Petri net representation of the Workflow Authorization Model provides us with good analysis tools for safety because the safety problem in the workflow authorization models can be made equivalent (with an appropriate PN representation) to the reachability problem in Petri nets. Thus, existing reachability analysis techniques, methods and results can be directly adapted to WAM. Therefore, in this section, we first provide a brief review of Petri nets. Then we show how Petri nets can be used as a modeling tool to represent WAM. We use a combination of two extended Petri net models – *colored* and *timed* Petri nets – to represent WAM as a Petri net.

3.1 Overview of Petri Nets

A *Petri Net* (PN) is a bipartite directed graph consisting of two kinds of nodes called *places* and *transitions* where arcs (edges) are either from a place to a transition or from a transition to a place. While drawing a PN, places are represented by circles and transitions by bars. A *marking* may be assigned to places. If a place p is marked with a value k, we say that p is marked with k *tokens*. Weights may be assigned to the edges of PN, however, in this paper we use only the ordinary PN where weights of the arcs are always equal to 1.

Definition 7. [16] A Petri net (PN) is a 5-tuple, $PN = (P, T, F, H, M)$ where
$P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places,
$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,

$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, and
$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.
$H = F \rightarrow \{1, 2, 3, \ldots\}$ is the weight of each arc,
$M = P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the marking. □

We use $m(p)$ to denote the marking of place p (or number of tokens in p), $f(p,t)$ to denote an arc from p to t and $f(t,p)$ to denote an arc from t to p.

A transition (place) has a certain number (possibly zero) of input and output places (transitions).

Definition 8. [16] Given a PN, the input and output set of transitions (places) for each place p_i (t_i) are defined as,
the set of input transitions of p_i, denoted $\bullet p_i = \{t_j | f(t_j, p_i) \in F\}$
the set of output transitions of p_i, denoted $p_i \bullet = \{t_j | f(p_i, t_j) \in F\}$, and
the input and output set of places for each transition t_i are defined as,
the set of input places of t_i, denoted $\bullet t_i = \{p_j | f(p_j, t_i) \in F\}$
the set of output places of t_i, denoted $t_i \bullet = \{p_j | f(t_i, p_j) \in F\}$. □

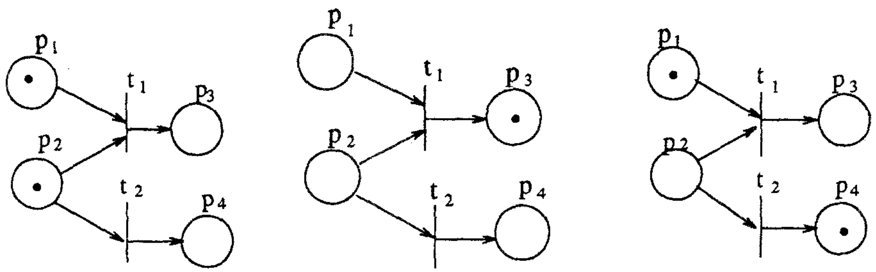
At any time a transition is either *enabled* or *disabled*. A transition t_i is enabled if each place in its input set $\bullet t_i$ has at least one token. An enabled transition can fire. In order to simulate the dynamic behavior of a system, a marking in a PN is changed when a transition fires. Firing of t_i removes the token from each place in $\bullet t_i$, and deposits one into each place in $t_i \bullet$. The consequence of firing a transition results in a change from original marking M to a new marking M'. For the sake of simplicity, we assume firing of a transition is an instantaneous event. The firing rules can be formally stated as follows:

Definition 9.

1. A transition t_i is said to be enabled if $\forall p_j \in \bullet t_i$, $(m(p_j) > 0)$. An enabled transition may fire.
2. Firing a transition t_i results in a new marking M' as follows: $\forall p_j \in \bullet t_i$, and $\forall p_k \in t_i \bullet$, $m'(p_j) = m(p_j) - 1 \wedge m'(p_k) = m(p_k) + 1$ □

Example 3. Figure 2 shows an example of a simple PN in which more than one firing sequence can be generated. It comprises of four places p_1, p_2, p_3, and p_4, and two transitions t_1 and t_2. The input and output sets of the places and transitions are as follows: $\bullet t_1 = \{p_1, p_2\}$, $\bullet t_2 = \{p_2\}$, $t_1 \bullet = \{p_3\}$, $t_2 \bullet = \{p_4\}$, $\bullet p_3 = \{t_1\}$, $\bullet p_4 = \{t_2\}$, $p_1 \bullet = \{t_1\}$, and $p_2 \bullet = \{t_1, t_2\}$.

The initial state of the PN is shown in figure 2(a) where p_1 and p_2 are both marked with one token each. Since both places in the input set of t_1 are marked (i.e., both $m(p_1)$, $m(p_2) > 0$), t_1 is enabled. Similarly, t_2 is also enabled as $m(p_2) > 0$. Although both t_1 and t_2 are enabled, firing one of them will disable the other. Thus, this net will result in two different firing sequences. Suppose t_1 fires first, it results in a new marking where the tokens from p_1 and p_2 are removed and a token is placed in p_3, as shown in figure 2(b). Since p_2 becomes empty, this disables t_2. The second firing sequence would result in by firing t_2



(a) Initial marking (b) if t_1 fires first (c) if t_2 fires first

Fig. 2. An example PN

first. It removes one token from p_2 and deposits one into p_4 thus resulting in a marking as depicted in figure 2(c). Since p_2 is empty, t_1 is disabled. Because there are no more transitions to fire after firing either t_1 or t_2, the PN stops (said to be *not live*).

Definition 10. A marking M is said to be *reachable* from a marking M_0 if there exists a sequence of firings that transforms M_0 to M. □

Reachability is a fundamental property for studying the dynamic properties of any system. It has been shown [12] that the reachability problem is decidable although it takes at least exponential space and time.

3.2 Petri Net Representation of WAM

We use a combination of timed and colored Petri nets [23, 15, 24] for representing WAM. The definition for colored and timed Petri nets is as follows:

Definition 11. A Colored + Timed Petri Net (CTPN) [10, 7][6] is a tuple $CTPN = (PN, \Sigma, C, E, D, IN)$ where
(i) $PN = (P, T, F, M)$ is an ordinary Petri net.
(ii) A finite set of colors or types, called color sets $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$
(iii) C is a color function such that $C(p) \in \Sigma_{MS}$,[7] and $\forall y \in m(p), C(y) \in \Sigma$
(iv) E, the arc set, defined from F into a set such that:
$\forall f(p,t), f(t,p) \in F, E_f \in C(p)_{MS}$
(v) D is a delay function, $D : P \rightarrow \mathcal{T}$
(vi) IN is an interval function such that $IN(t) = [\tau_l(t), \tau_u(t)] \in \{\mathcal{T} \times \mathcal{T} | \tau_l(t) \leq \tau_u(t)\}$. □

[6] Although we use the term "timed," our timed PN is different from the traditional timed PN.

[7] Σ_{MS} represents a multi-set or a bag over Σ. For example, given a set $\Sigma = \{a, b, \ldots\}$, the multi-sets $a, a + b, a + 2b$ are members of Σ_{MS}.

We represent a token in place p as (v, x) where $v \in C(m(p))$ represents the color of the token and x represents its timestamp such that $x \in \mathcal{T}$. Whenever a token moves from one place to another through a fired transition, its timestamp is modified as the firing time of the transition.

The above definition dictates that each token has a color (or type) which is defined in the color set Σ. Each place has a color set (i.e., denoted as $C(p)$) attached to it which specifies the set of allowable colors of the tokens to enter the place. For a token to reside in a place, it must satisfy that the color of token is a member in the color set of the place. Each arc $f(p,t)$ or $f(t,p)$ is associated with a color set such that this set is contained in the multi-set of $C(p)$.

A transition t is enabled only if all of its input places p contain at least as many tokens of the type as that specified in the arc set $E_{f(p,t)}$ of the corresponding $f(p,t)$. An enabled transition fires after the delay $D(p) = d$ specified in its input place. t fires only if this time falls within the specified time interval $IN(t)$. Both the time interval and delay can be specified as variables instead of fixed values. When more than one input place exists, the transition fires after the maximum delay of all the input places has elapsed.[8] Upon firing, a transition t consumes as many tokens of colors from each of its input places p as those specified in the corresponding $E_{f(p,t)}$ and deposits as many tokens with specified colors into each output place p as those specified in the corresponding $E_{f(t,p)}$. That is, the arc set of $f(p,t)$ specifies the number of tokens of specified colors to be removed from p when t fires, and the arc set $f(t,p)$ specifies the number of tokens of specified colors to be inserted into p when t fires.

Marking of a place $m(p)$ is expressed as a list of tokens with respect to distinctive colors (e.g., $m(p) = 3\langle g\rangle, 2\langle r\rangle$). A transformation of colors may occur during firing of a transition. The firing of a transition is determined by the firing rules and the transformation by the arc set E.

The firing rules can be formally stated as follows:

Definition 12. Given a transition t_i such that $IN(t_i) = [\tau_l(t), \tau_u(t)]$, $\forall p_j \in \bullet t_i$ and $\forall p_k \in t_i \bullet$,

1. t_i is said to be *enabled* if $E_{(p_j,t_i)} \subseteq m(p_j)$ and $\max\{x + D(p_j)$ where $x = \max\{x_j | x_j$ is the timestamp of all $m(p_j)\}\}$ is within $IN(t_i)$. An enabled transition may fire.
2. Suppose t_i fires at τ_i. Firing of t_i results in a new marking M' as follows: $m'(p_k) = m(p_k) + E_{f(t_i,p_k)}$[9] and the timestamp of each element in $m'(p_k)$ is τ_i. □

We now illustrate the working of CTPN with a simple example. Assume there are two places p_1 and p_2 and a transition t_1 as shown in figure 3(a) such that the color sets of p_1 and p_2 are $C(p_1) = \{\langle a\rangle, \langle b\rangle, \langle c\rangle\}$ and $C(p_2) = \{\langle a\rangle, \langle c\rangle, \langle d\rangle\}$,

[8] If two or more transitions have the same input set, then more than one transition is enabled. In that case, only one transition fires. Selection of the firing transition among the many enabled transitions is determined non-deterministically.

[9] Note that these two are bags but not sets.

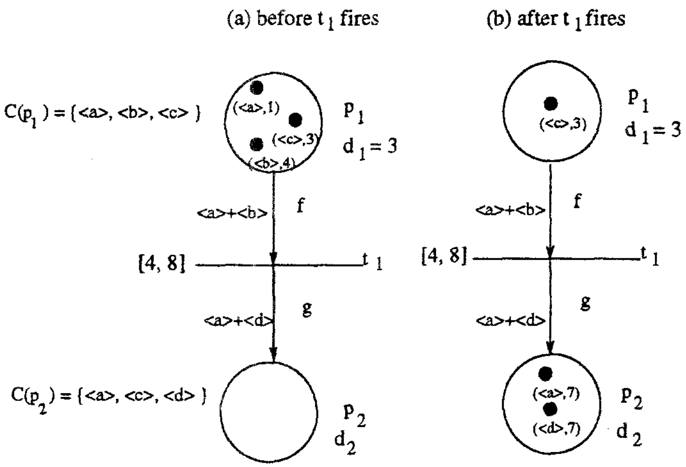


Fig. 3. An example showing the working of CTPN

respectively. The delay associated with p_1, i.e., $d_1 = 3$. The time interval of t_1, $IN(t_1) = [4, 8]$. Suppose p_1 is initially marked with three tokens: one of each color $(\langle a\rangle, 1)$, $(\langle b\rangle, 3)$ and $(\langle c\rangle, 4)$. The arc set associated with the two arcs f and g are $E_f = \langle a\rangle + \langle b\rangle$ and $E_g = \langle a\rangle + \langle d\rangle$. The corresponding CTPN is shown in figure 3(a). Transition t_1 is enabled at time 7 because the maximum timestamp of all the tokens in p_1 is 4 and the time delay of p_1 is 3. Since $E_f \subset C(p_1)$ and and the enabled time is within $IN(t_i)$, t_1 is enabled. When t_1 fires, one token of each color $\langle a\rangle$ and $\langle b\rangle$ are removed from p_1, and according to E_g one token of each color $\langle a\rangle$ and $\langle d\rangle$ with timestamp $= 7$ are deposited in p_2. The resulting CTPN is shown in figure 3(b).

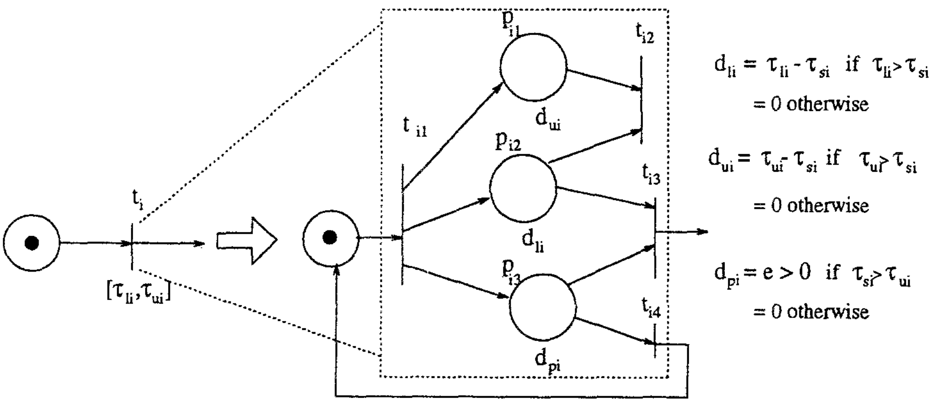


Fig. 4. A CTPN with temporal constraints

We have represented the temporal constraint by attaching t with the time interval $[\tau_l(t), \tau_u(t)]$ in the constraint as a gate on transition t. This means, t is enabled only during this time interval and is disabled during all other times even if tokens are present in its input places. A detailed implementation of the temporal constraint for each transition t is shown in figure 4. Note that the delays d_{l_i} and d_{u_i} are not predefined constants but vary depending on the enabled time (or token arrival time) τ_{a_i} as follows: If $\tau_{l_i} > \tau_{a_i}$, then $d_{l_i} = \tau_{l_i} - \tau_{a_i}$, otherwise $d_{l_i} = 0$. If $\tau_{u_i} > \tau_{a_i}$, then $d_{u_i} = \tau_{u_i} - \tau_{a_i}$, otherwise $d_{u_i} = 0$. If $\tau_{a_i} > \tau_{u_i}$, then $d_{p_i} = \epsilon$ (where ϵ is a small positive value). It works as follows: when t_{i_1} fires, p_{i_1}, p_{i_2} and p_{i_3} are all marked. If $\tau_{a_i} \leq \tau_{l_i}$, delay d_{l_i} ensures that t_{i_3} can be enabled no sooner than τ_{l_i} thereby guaranteeing the lower boundary of the constraint. If $\tau_{l_i} \leq \tau_{a_i} \leq \tau_{u_i}$, d_{l_i} becomes 0 which allows t_{i_3} to be enabled and therefore fire immediately. When $\tau_{a_i} > \tau_{u_i}$, both d_{l_i}, d_{u_i} are 0 and $d_{p_i} = \epsilon$. Thus t_{i_2} is enabled and therefore fires, disabling t_{i_3}. Since p_{i_2} is no longer marked, t_{i_3} cannot fire, thus ensuring that authorizations are never granted if the object arrives after the upper bound on the constraint has elapsed.

3.3 CTPN representation of WAM

In the Petri net representation of WAM, we use two distinct color sets Ω and Λ such that $\Omega \cup \Lambda = \Sigma$ and $\Omega \cap \Lambda = \emptyset$, where we use Ω to denote the types of objects and Λ to denote the different privileges.

To represent WAM as a CTPN, we use the following mapping:

1. Two transitions t_s and t_f to represent the beginning and ending of a task, and the time at which they fire denote τ_s and τ_f, respectively.
2. A time interval $IN(t)$ at each t_s and t_f to represent the specified time constraint of the task ($[\tau_l, \tau_u]$).
3. A place to represent the execution state of the task w (depicted as a circle)
4. Different colored tokens to represent different types of objects $(\omega_1, \omega_2 \ldots)$, i.e., $\Omega = \Gamma$.
5. A place to represent each subject (denoted as a square in the diagram), i.e., s in $AT(w)$.[10]
6. Different colored tokens to represent different types of object-privilege pairs $(\lambda_1, \lambda_2 \ldots)$, i.e., $\Lambda = \Gamma \times PR$.
7. A color set associated with circles (squares) to denote the allowed type of objects (object-privilege pairs)
8. An arc set of $f(p, t)$ (where p is a circle) to represent the type of input objects to be sent to a task for execution.
9. An arc set of $f(t, p)$ (where p is a circle) to represent the type of output objects of the task.

[10] If a task is associated with multiple AT's with different subjects, then each subject has to be represented as a different square. On the other hand, if a single subject is involved in number of AT's for that task, then the arc function of the input arc reflects this.

10. An arc set of $f(p,t)$ (where p is a square) to represent the privileges (represented as object-privilege pairs) to be granted when a task starts.
11. An arc set of $f(t,p)$ (where p is a square) to represent the privileges to be revoked when a task completes its execution.
12. A delay (d) associated with the place to represent the execution time of the task. Note that d is associated with only places representing tasks (circles) but not subjects (squares).
13. A token (v,x) to represent the movement of objects (privileges) to and from the tasks (subjects) where v is the color of the token representing the type of the object (object-privilege pair), and x the timestamp representing the arrival time of the object τ_a.

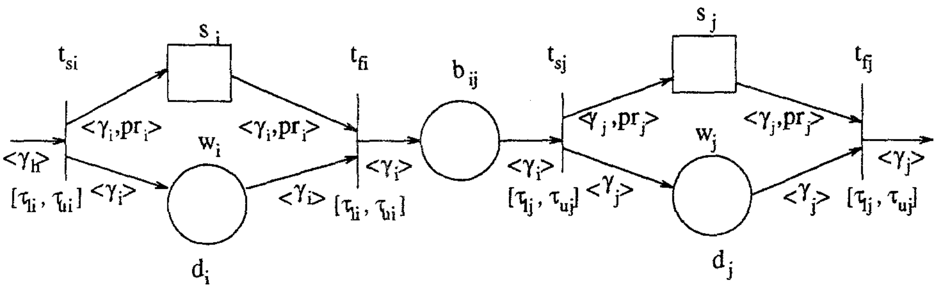
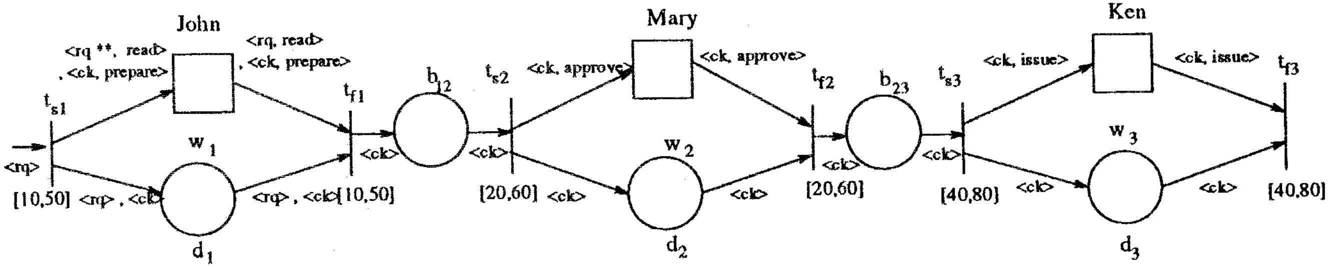


Fig. 5. A CTPN representation of WAM for a Workflow with two tasks

Figure 5 shows a CTPN representation of the authorization model for a workflow consisting of two tasks w_i and w_j. The execution state of w_i and w_j are represented as two places (circles), and the subjects authorized to execute these two tasks s_i and s_j are represented as another two places (squares). t_{s_i} fires when an object of type γ_i arrives, thus starting w_i. A token of color γ_i is placed in w_i and another token $\langle\gamma_i, pr_i\rangle$ is placed in s_i. Thus privilege is granted to s_i on the object only at this point. After d_i, the task completes its execution thus firing t_{f_i}. This removes the objects from w_i and place them in another place b_{ij} since both $E_{f(w_i,t_{f_i})}$ and $E_{f(t_{f_i},b_{ij})}$ is $\{\langle\gamma_i\rangle\}$. Here b_{ij} represents the state after w_i finishes but before w_j starts. Firing of t_{f_i} also removes the privilege $\langle\gamma_i, pr_i\rangle$ from s_i, but does not deposit any token in b_{ij}.

The CTPN of the WAM for the workflow in example 2 is shown in figure 6. Figure 7 shows the state when w_1 starts execution. It shows objects and the privileges as tokens in w_i and "John," respectively. The two tokens in each place correspond to request and check. Figures 8 and 9 depict the states when w_1 finishes execution and w_2 starts execution, respectively.



$AT_1(w_1)$ = (John, (request, -), read)

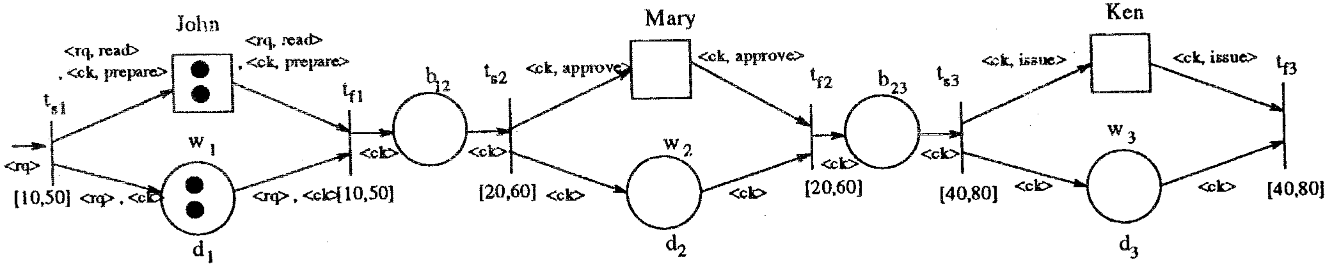
$AT_2(w_1)$ = (John, (check, -), prepare)

$AT(w_2)$ = (Mary, (check, -), approve)

$AT(w_3)$ = (Ken, (check, -), issue)

** Remark: For simplicity of notation , the <request> and <check> types are subsituted with <rq> and <ck>, respectively

Fig. 6. CTPN for the WAM of the Check Processing workflow



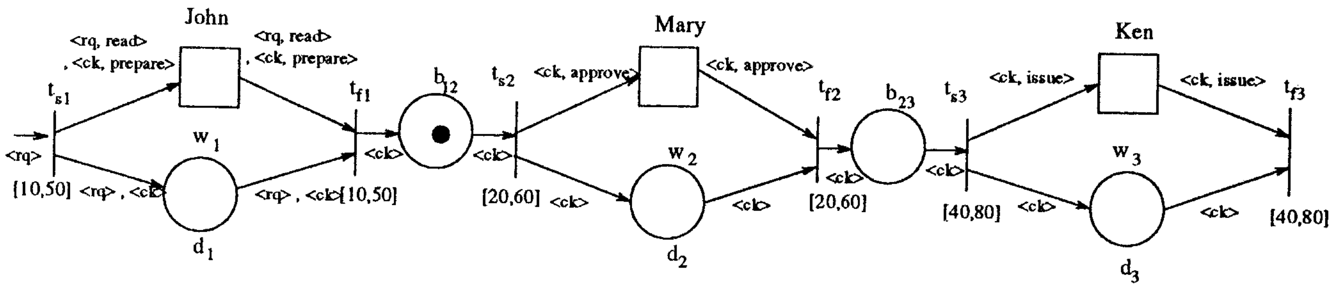
A_{11}= (John, rq1, read, [40,50])

A_{12}= (John, ck023, prepare, [40,50])

$AT(w_2)$ = (Mary, (check, -), approve)

$AT(w_3)$ = (Ken, (check, -), issue)

Fig. 7. When w_1 starts



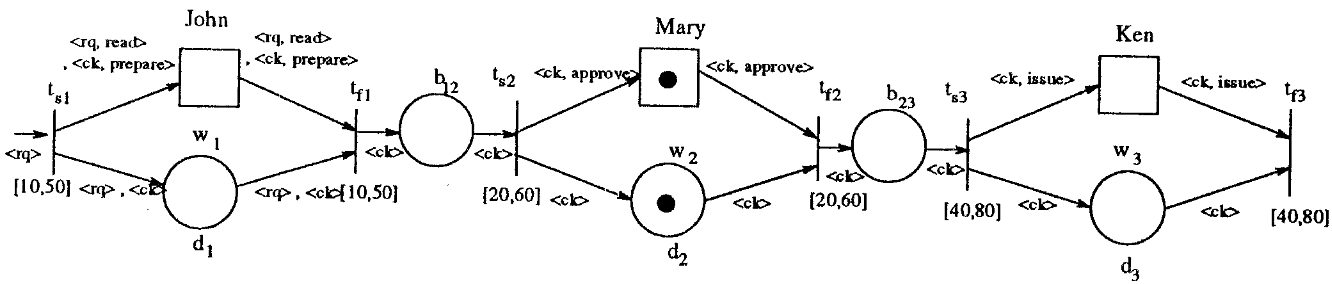
A_{11} = (John, rq1, read, [40,47])

A_{12} = (John, ck023, prepare, [40,47])

$AT(w_2)$ = (Mary, (check, -), approve)

$AT(w_3)$ = (Ken, (check, -), issue)

Fig. 8. When w_1 finishes



A_{11} = (John, rq1, read, [40,47])

A_{12} = (John, ck023, prepare, [40,47])

A_2 = (Mary, ck023, approve, [47,60])

$AT(w_3)$ = (Ken, (check, -), issue)

Fig. 9. When w_2 starts

4 Extensions to the Workflow Authorization Model

In the WAM proposed in section 2 and its Petri net representation presented in section 3, we have not incorporated all the desirable features of WAM described in section 1. In this section, we will provide a brief explanation of how role-based authorizations and separation of duties can be incorporated into our model.

Most commercial Workflow Management Systems (WFMSs) such as Lotus Notes and Action Workflow enforce security based on organizational roles [14, 13, 9]. Privilege to perform a task is assigned to an organizational role rather than to human users, and human users in turn are authorized to assume prespecified roles. It is particularly beneficial in workflow environments to facilitate dynamic load balancing when a task can be performed by several individuals.

Note that the subject in the AT's can be specified in terms of roles [19, 21] by replacing s with R in AT as follows:

$$AT = ((R, -)(\gamma, -), pr, [\tau_l, \tau_u])^{11}$$

For example, considering example 2 once again, the corresponding $AT's$ are modified as:

$AT_1(w_1)$ = (clerk, (request,-),read)
$AT_2(w_1)$ = (clerk, (check,-), prepare)
$AT(w_2)$= (supervisor, (check,-), approve)
$AT(w_3)$= (clerk, (check,-), issue)

The authorization derivation rule to derive A from AT need to be modified as follows.

Definition 13 Authorization Derivation Rule with Roles. Given an authorization template $AT(w_i) = (R_i, (\gamma_i, -), pr_i)$ of task $w_i = (OP_i, \Gamma_{IN_i}, \Gamma_{OUT_i}, [\tau_{l_i}, \tau_{u_i}])$, an authorization $A_i = (s_i, o_i, pr_i, [\tau_{b_i}, \tau_{e_i}])$ is derived as follows:
Grant Rule: Suppose object $o_i \in \Gamma_{IN_i}$ is sent to w_i at τ_{a_i} to start w_i. Let the starting time of w_i be τ_{s_i}.
If $\tau_{a_i} \leq \tau_{u_i}$, then $s_i \in R_i$, $pr_i \leftarrow pr(AT)$, $\tau_{e_i} \leftarrow \tau_{u_i}$, and
(if $\tau_{a_i} \leq \tau_{l_i}$ then $\tau_{b_i} \leftarrow \tau_{l_i}$; otherwise $\tau_{b_i} \leftarrow \tau_{a_i}$.
Revoke Rule: Suppose w_i ends at τ_{f_i} at which point o_i leaves w_i.
If $\tau_{f_i} \leq \tau_{u_i}$, then $\tau_{e_i} \leftarrow \tau_{f_i}$. □

This authorization rule is similar to that in definition 6 except that we select a subject from the set of subjects playing the specific role while assigning the authorizations.

Separation of duties can be incorporated by including the identity of the place (i.e. subject) to the token. That is, a token is of the form $(\langle p, v\rangle, x)$ where p is the place v refers to the object type and x is the current timestamp.

[11] Here the notion of $(R, -)$ is similar to that of object hole. The actual authorization is derived when this is filled by a subject.

5 Conclusions and Future work

In this paper, we have presented an authorization model for workflow environments that is capable of synchronizing the authorization flow along with the workflow. Our model is also capable of ensuring temporal constraints on tasks. We have provided an implementation model based on Petri nets. We have also shown how to incorporate authorizations that can be assigned to organizational roles rather than to subjects and how separation of duties can be incorporated. As part of future work, we intend to implement WAM and conduct the safety analysis. Moreover, in this paper, we have not considered the various dependencies among the tasks within a workflow. A complete model should take into account these dependencies as well. We intend to combine the Petri net models developed in [1] into the proposed CTPN.

References

1. Vijayalakshmi Atluri and Wei-Kuang Huang. An extended petri net model for supporting workflows in a multilevel secure environment. In *Proc. of the 10th IFIP WG 11.3 Workshop on Database Security*, July 1996.
2. Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati. A temporal access control mechanism for database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):67–80, 1996.
3. Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. Authorizations in relational database management systems. In *Proc. First ACM Conference on Computer and Communications Security*, Fairfax, VA, November 1993.
4. Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. High assurance discretionary access control for object bases. In *Proc. First ACM Conference on Computer and Communications Security*, Fairfax, VA, November 1993.
5. J. Biskup and C. Eckert. About the enforcement of state dependent security specifications. In *Proc. of the 7th IFIP WG 11.3 Workshop on Database Security*, pages 3–17, August 1993.
6. David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proc. IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, California, April 1987.
7. Rene David and Hassane Alla. *Petri Nets and Grafcet - Tools for modeling discrete event systems*. Prentice Hall, 1992.
8. E. B. Fernandez, E. Gudes, and H. Song. A security model for object-oriented databases. *Proc. IEEE Symposium on Security and Privacy*, pages 110–115, May 1989.
9. Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, pages 119–153, 1995.
10. K. Jensen. Colour petri nets: A high level language for system design and analysis. In K.Jensen and G. Rozenberg, editors, *High-level Petri Nets - Theory and Application*, pages 44–119. Springer-Verlag, Lecture Notes in Computer Science, 1991.

11. D. Johnscher and K.R. Dittrich. Argos - A configurable access control system for interoperable environments. In *Proc. of the 9th IFIP WG 11.3 Workshop on Database Security*, pages 39–63, August 1995.
12. S. R. Kosaraju. Decidability and reachability in vector addition systems. In *Proc. of the 14th ACM Symposium on Theory of Computing*, pages 267–281, May 1982.
13. Lotus Corporation. *Lotus Notes Administrator's Reference Manual, Release 4*, 1996.
14. Raul Medina-Mora, Harry K.T. Wong, and Pablo Flores. ActionWorkflow^{t}m as the enterprise integration technology. *Bulletin of IEEE Technical Committee on Data Engineering*, 16(2):49–52, 1993.
15. S. Morasca, M.Pezzè, and M. Trubian. Timed high-level nets. *Journal of Real-Time Systems*, 3:165 – 189, 1991.
16. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
17. F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans. on Database Systems*, 16(1):88–131, March 1991.
18. Pierangela Samarati, Paul Ammann, and Sushil Jajodia. Propagation of authorizations in distributed database systems. In *Proc. Second ACM Conference on Computer and Communications Security*, Fairfax, VA, November 1994.
19. Ravi S. Sandhu. Transaction control expressions for separation of duties. In *Fourth Computer Security Applications Conference*, pages 282–286, 1988.
20. Ravi S. Sandhu. Separation of duties in computerized information systems. In Sushil Jajodia and Carl Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 179–189. North Holland, 1991.
21. Ravi S. Sandhu. Role-based access control models. *IEEE Computer*, pages 38–47, February 1996.
22. Ravi S. Sandhu and Gurpreet S. Suri. Non-monotonic transformation of access rights. In *Proc. IEEE Symposium on Security and Privacy*, pages 148–161, Oakland, California, May 1992.
23. W.M.P van der Aalst. Interval timed coloured petri nets and their analysis. In *Application and Theory of Petri Nets 1993, Proc. 14th International Conference*, volume 691, pages 453–472, Chicago, (USA), 1993. Springer-Verlag, Lecture Notes in Computer Science.
24. K.M. van Hee, L.J. Somers, and M. Voorhoeve. Executable specifications for distributed information systems. In E.D. Falkenberg and P. Lindgreen, editors, *Proc. of the IFIP TC 8/WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis*, volume 691, pages 139–156, Namur, (Belgium), 1989. Elsevier Science Publishers, Amsterdam.
25. Thomas Y.C. Woo and Simon S. Lam. Authorization in distributed systems: A formal approach. In *Proc. IEEE Symposium on Security and Privacy*, pages 33–50, Oakland, California, May 1992.
26. William A. Wulf, Roy Levin, and Samuel P. Harbison. *HYDRA/C.mmp, An Experimental Computer System*. McGraw-Hill, 1981.