Security for Workflow Systems

Lê Thanh Dũng
Võ Kim Huân
Trần Thùy Nhiên
Lưu Huỳnh Trung Hiếu
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
1. Introduction

- **Definition**
  - A workflow is defined as a set of coordinated activities that achieves a common business objective.
  - A workflow management system (WFMS) is a system that supports process specification, enactment, monitoring, coordination, and administration of workflow process through the execution of software, whose order of execution is based on the workflow logic.
1. Introduction

Definition

- Workflow management aims at modeling and controlling the execution of business processes involving a combination of manual and automated activities in an organization.
1. Introduction

- Example Workflow
1. Introduction

- Why workflow systems are necessary?

- Organizations are seeking ways to effectively integrate and automate their business processes.
- The business policy is still often hard-coded in applications.
- Workflows are not necessarily bound to a single organization, but may span multiple organizations.
1. Introduction

- Advantages of workflow systems
  - Separate the business policy from the business applications to enhance flexibility and maintainability of business process reengineering.
  - Support resource allocation and dynamically adapting to workload changes.
1. Introduction

2. Security requirements in workflow systems

3. Workflow authorization model

4. Separation of duty

5. Delegation of authority

6. Conflict of interest

7. Workflow safety analysis

8. Open issues
2. Security requirements in workflow systems

- Authorization
- Separation of duty
- Delegation
- Conflict of Interest
- Safety analysis
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
3. Workflow authorization model

- Workflow Authorization Model (WAM): specify 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.

- Authorization Template (AT) is comprised of the static parameters of the authorization that can be defined during the design of the workflow.
3. Workflow authorization model

- A new authorization is granted to an executing agent only when an object hole is filled with an appropriate object.
- Besides, users are assigned to appropriate roles based on their qualifications.
3. Workflow authorization model

AT(T1)=(employee, (claim\textsuperscript{o}), prepare)  
AT(T2)=(supervisor, (claim\textsuperscript{o}), approve)  
AT(T3)=(clerk, (claim\textsuperscript{o}), issue)
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. **Separation of duty**
5. Delegation of authority
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
Workflow

- Set of coordinated activities (tasks) to achieve a common business objective
- Used in numerous application domains
  - Office Automation
  - Finance and banking
  - Web-services
  - Healthcare, …
- Tasks are carried out by agents according to a set of organizational rules
- Schema
  - The specification of tasks and control flow requirements
- Workflow Instance
  - Generated when a workflow is executed
  - Tasks are assigned to users based on roles and the other (security) policies
Authorization Constraints

- Purpose is to mitigate fraud and collusion among users
- In a workflow environment → Forbid users or roles from performing certain task(s)

Check Processing Workflow

“prepare check” and “issue check” should not be executed by the same user
Authorization Constraints

- Intra-instance Constraints
  - specified on a workflow schema and therefore apply to a single instance
  - Static: can be enforced at the time of schema specification
  - Dynamic: can only be enforced at run-time

- Inter-instance Constraints
  - specified on instances rather than on the workflow schema
  - Can be specified on multiple instances of the same workflow that can only be enforced at run-time
  - can be specified on the history of instances and therefore are not necessarily limited to one workflow
  - Other purposes
    - workload and resource distribution
Need for Inter-instance Constraints

- **Prevent collusion**
  - Approve each others loans!!

- **Prevent fraud over time**
  - Approve large amount of loans in a day

- **Binding**
  - Exploit the experience: An individual should be assigned to a task because of previous experience with conducting that task
Example Bank Loan Approval Process

- Prevent collusion
  - Two critical tasks can only be performed by the same group of people three times.
  - There should not exist a group of n (say 2) users represented by G and m instances (say 4) such that if \( t_1 \) is executed by a user in G, \( t_2 \) is executed by the remaining users in G in more than m instances.
Example Bank Loan Approval Process

- **Prevent fraud over time:**
  - A user is not allowed to do $t_2$ if the total loan amount per day exceeds $1M$.
  - There should not exist more than one instance of $W$ such that the input parameters (say loan customer) is the same and the loan amounts sum up to $100K$ during a period of one month.
  - There should not exist more than 4 instances of $W$ such that a specific input parameter (say loan customer) is the same and the user executing $t_2$ is also the same.
  - A user is not allowed to execute more than 100 tasks (of any workflow) in a day.
Our Goal

- Assign users/roles to tasks
  - By adhering to both intra-instance and inter-instance authorization constraints
  - Identify/rectify certain anomalies
    - Inconsistency
    - Overlapping anomaly
    - Depletion anomaly
Related Work

- Bertino, Ferrari, Atluri (BFA) Model, RBAC 97, TISSEC 1999
  - enables specification of separation of duties and proposes methodologies to analyze their consistency
  - not capable of distinguishing one instance from the other, and therefore cannot model inter-instance constraints
  - Extend the specification language of the BFA model to handle inter-instance constraints as well as handle conditions on temporal and workflow parameters

- Crampton et al., SACMAT 2005 and IEEE Computer Security Foundations Workshop, 2004
  - Alternative authorization model to which our inter-instance constraint ideas could also apply

- Botha and Eloff, IBM Systems Journal 2001
  - Introduced concept of constraining authorization based on conflicting users, tasks, roles, permissions, but does not handle inter-instance constraints

- Atluri, Bertino, Ferrari and Mazzoleni, IFIP03
  - Considered delegation rules (in addition to the SOD constraints)

- Atluri, Warner, SACMAT 05
  - Considered conditional delegation, role activation and workflow dependency requirements (in addition to the above)
Constraint Specification Language

- Constants
  - Temporal and workflow constants, users, roles, tasks, task instances
- Variables
  - Temporal variables, workflow input and output variables, role and user variables, task and task instance variables
- Predicates
  - External Specification Predicates
  - Workflow Specification Predicates
  - Enforcement Predicates
  - Status Predicates
  - Conditional Predicates
  - Comparison Predicates
- Rules
Predicates

- **External Specification Predicates**
  - related($u_i, u_j$)
  - partner-of ($u_i, u_j$)
  - same-group($u_i, u_j$)

- **Workflow Specification Predicate**
  - role($R_i, t_j$)
  - user($u_i, t_j$)
  - belong($u_i, R_j$)
  - belong($u_i, R_j$)
  - ..
  - critical-task-pair($t_i, t_j$)
Predicates

- **Enforcement Predicates**
  - cannot_do\textsubscript{u}(u\textsubscript{i}, t\textsubscript{j}), cannot_do\textsubscript{u}(u\textsubscript{i}, t\textsubscript{j}^k)
  - cannot_do\textsubscript{R}(R\textsubscript{i}, t\textsubscript{j}), cannot_do\textsubscript{R}(R\textsubscript{i}, t\textsubscript{j}^k)
  - must_execute\textsubscript{R}(R\textsubscript{B}, t\textsubscript{2}), must_execute\textsubscript{R}(R\textsubscript{B}, t\textsubscript{2}^k)
  - must_execute\textsubscript{R}(R\textsubscript{B}, t\textsubscript{2}), must_execute\textsubscript{R}(R\textsubscript{B}, t\textsubscript{2}^k)
  - abort(W), abort(t\textsubscript{j}^k)

- **Status Predicates**
  - executed\textsubscript{u}(u\textsubscript{i}, t\textsubscript{j}^k), executed\textsubscript{R}(R\textsubscript{i}, t\textsubscript{j}^k)
  - assigned\textsubscript{u}(u\textsubscript{i}, t\textsubscript{j}^k)
  - succeeded (t\textsubscript{ij}^k)
  - aborted (t\textsubscript{ij}^k)
  - collaborator(u\textsubscript{i}, u\textsubscript{j})
Predicates

- Conditional Predicates
  - count and \( \text{count}_{nv} \)
  - min, max, sum and avg
  - \( \text{timestamp}(t_i^k, ts) \)
  - \( \text{time-interval}(t_i^k, t_j^l) \)

- Comparison Predicates
A user is not allowed to do $t_2$ if the total loan amount per day exceeds $1M$.

$$\text{cannot}_{do}^u(u_i, t_2^k) \leftarrow \forall i, \text{time-interval}(t_2^k, t_2^i) < \text{“24 hours"}, \sum(i, (\text{output}(t_2^i).\text{loan-amount}), n)), n > 1M$$

**Predicates**
- **cannot_do**: refers to a task in an instance of a workflow
- **Predicate provides time between two instances of a task.**
- **Parameters of workflow**
- **Used in specifying constraints**
Constraint specification

- There should not exist more than 4 instances of \( W \) such that the loan customer is the same and the user executing \( t_2 \) is also the same.

\[
\text{cannot}_u(u_j, t_2^k) \leftarrow \forall i, \; \text{count}_{n v}(u_j, \text{executed}_u(u_j, t_2^i), n), \; n > 4, \; \text{input}(t_2^i).\text{loan-customer} = \text{input}(t_2^k).\text{loan-customer}
\]
Constraint specification

- There should not exist a group of n (say 2) users represented by G and m instances (say 4) such that if t1 is executed by a user in G, t2 is executed by the remaining users in G in more than m instances.

\[
cannot_{do}^{u}(u_{i}, t_{2}) \leftarrow ((count (collaborator(u_{i}, u_{j}), n), n = 3, \\
executed^{u}(u_{j}, t_{1}));
\]

\[
cannot_{do}^{u}(u_{i}, t_{1}) \leftarrow ((count (collaborator(u_{i}, u_{j}), n), n = 3, execut^{u}(u_{j}, \\
t_{2});
\]

Indicates when users u and u have executed tasks identified as being critical task pairs.
The set of all constraints: CB
More than one rule may be used to specify a constraint
We can prove that CB is a stratified normal program
Constraint Anomalies

- Inconsistency
- Overlapping anomaly
- Depletion anomaly
Inconsistency

- Results from a conflict in the constraint base.
  - Due to one set of rules, a user must perform a task
  - Due to another set of rules, a user is denied authority to perform the same task.

- There exist some users that are allowed to do a task as well as not allowed to do a task
  - Checked statically by calculating the sets of roles/users that must be prevented from executing task $t_i$ and the set of roles/users permitted to execute task $t_i$ and comparing them
    - $\text{Denied}_\text{Users}^S$, $\text{Denied}_\text{Roles}^S$, $\text{Permitted}_\text{Users}^S$, and $\text{Permitted}_\text{Roles}^S$
  - If constraints concerning task instance $t_i^k$ depend upon run-time parameters or conditions, new sets are calculated at run-time
    - $\text{Denied}_\text{Users}^R$, $\text{Denied}_\text{Roles}^R$, $\text{Permitted}_\text{Users}^R$, and $\text{Permitted}_\text{Roles}^R$
Overlapping Anomaly

- Arises when the conditions for one constraint overlap with the conditions for another constraint
  - $[30,40][20,50]; x>50, x>100$
  - Can be handled before workflow is instantiated by matching the conditions
  - If the result of the conflicting constraints is the same, the less restrictive condition can be tightened.
  - If the result of the conflicting constraints is opposite, additionally we have an inconsistency and the conditions must be made not to overlap
    - cannot_do[30,40], must_execute[20,50]
Depletion Anomaly

- Arises when due to previous assignments to tasks and workflow instances, there is no one with the rights to complete particular tasks in current instances.
  - Static depletion anomaly is easily detected when the sets of permitted users/roles contain no members.
    - Empty Permitted_Users$^S$ and Permitted_Roles$^S$
    - can be rectified by modifying constraints.
  - Conditional depletion anomaly occurs during runtime when due to conditions and previous assignments, there is temporarily or permanently no one who can be assigned to a task
    - Empty Permitted_Users$^R$ and Permitted_Roles$^R$
**Anomaly Detection and Task Assignment**

**Workflow Schema Defined**

- **Step 1 – Static Enhancement of CB When Workflow Schema is Defined**
- **Step 2 – Static Inconsistency Identification and Analysis**
- **Step 3 – Run-Time Inconsistency Identification and Analysis**
- **Step 4 – Run-time Updating of CB if Conditions Warrant it.**

**Task Instance**

**Step 1** - Performed when workflow is defined.
**Step 2** – Performed when workflow is defined and when new rules are added.
**Step 3** – Performed whenever a task instance is initiated if there are constraint conditions met by the parameters of the instance.
**Step 4** – Performed whenever assignments made will constrain assignments for future instances of the workflow.
Step 1 – Static Enhancement of CB

- Derivation Rules are Applied:
  - executed\( (u_i, t_{j^k}) \leftarrow assigned(u_i, t_{j^k}), succeeded(t_{j^k}) \)
  - collaborator\( (u_i, u_j) \leftarrow critical-task-pair(t_{sj}, t_{sk}), \)
    \(executed(u_i, t_{sj}^m), executed(u_i, t_{sk}^m) \)
  - related\( (u_i, u_j) \leftarrow related(u_i, u_k), related(u_k, u_j), u_i \neq u_j \neq u_k \)
  - partner_of\( (u_i, u_j) \leftarrow partner_of(u_i, u_k), \)
    \(partner_of(u_k, u_j), u_i \neq u_j \neq u_k \)
  - critical-task-pair\( (t_{ij}, t_{ik}) \leftarrow critical-task-pair(t_{ij}, t_{il}), \)
    \(critical-task-pair(t_{il}, t_{ik}), t_{ij} \neq t_{il} \neq t_{ik} \)

- Overlapping anomalies are rectified
Step 2 – Static Anomaly Identification and Rectification

- Constraints that do not have run-time conditions (Static Constraints) are separated from those that do (Conditional Constraints).
- For each task, the static constraints are applied to calculate the following sets:
  - Denied_Users\(^S\), Denied_Roles\(^S\), Permitted_Users\(^S\), and Permitted_Roles\(^S\)
- If intersections between the denied and permitted user/role sets, the schema is inconsistent under any conditions and must be modified.
Step 3 – Run-time Anomaly Identification and Rectification

- Only occurs when a workflow task is instantiated that has conditional constraints
  - If it does not, assignments are made using the calculated sets Permitted_Users^S and Permitted_Roles^S.

- Real-time conditions are used to calculate Denied_Users^R, Denied_Roles^R, Permitted_Users^R, and Permitted_Roles^R.

- If inconsistencies or depletion anomalies are discovered, a system-specific rectification is made.

- Otherwise, assignments are made from the calculated sets Permitted_Users^R and Permitted_Roles^R.
Step 4 – Ongoing Update of CB

- As tasks complete their execute, new rules or modified rules are created when conditional constraints are met.
  - Added to make the computation of the sets of Denied_Users^R, Denied_Roles^R, Permitted_Users^R, and Permitted_Roles^R more efficient.
An Example

- Workflow: $T = \{t_1, t_2\}$
  - $t_1$: Review request and propose response.
  - $t_2$: Approve response
- Both tasks can generally be done by role $R_A$ consisting of members John, Lisa, Paul, Pam, and Sam.
- When the workflow concerns a high-valued customer, StarCo, a role $R_B$ whose members are Robert and Jane must perform task $t_2$. 
Step 1: By virtue of rules 1 and 2, the CB is enhanced:

6. related(Sam, Pam) ←

Step 2: Permitted_users^S(t_1) = \{John, Lisa, Paul, Pam, Sam\}, Permitted_roles^S(t_1) = \{R_A\}, Permitted_users^S(t_2) = \{John, Lisa, Paul, Pam, Sam\}, Permitted_roles^S(t_2) = \{R_A\}, Denied_users^S(t_1) = \emptyset, Denied_roles^S(t_1) = \emptyset, Denied_users^S(t_2) = \emptyset, Denied_roles^S(t_2) = \emptyset

No static anomalies.

No run-time conditions for t_1 so assignments are chosen from the static sets of permitted users and roles.
Updated CB

1. related(Sam, Paul) ← Sam and Paul have a relationship.
2. related(Pam, Robert) ← Pam and Robert have a relationship.
3. cannot_do(u_j, t_2^k) ← ∀i, count_{nv}(u_i, executed_u(u_j,t_2^i), n), n> 2
   A user can only execute two instances of task t_2.
4. cannot_do(u_j, t_2^k) ← executed_u(u_j,t_1^i), n), input(W_k).customer = input(W_i).customer, related(u_i, u_j)
   A user cannot execute task t_2 when the someone who has a relationship with them has performed task t_1 in any other instance for the same customer.
5. must_execute_R_{R_B}(t_2^k) ← input(W_k).customer = “StarCo”
   Someone from Role R_B must perform task t_2 when the customer is StarCo.
6. related(Sam, Pam) ←

Step 3: Workflow instance W_1 begun for customer StarCo.
Permitted_users^R(t_2^1) = \{Robert, Jane\},
Permitted_roles^R(t_2^1) = \{R_B\},
Denied_users^R(t_2^1) = \emptyset
Denied_roles^R(t_2^1) = \emptyset

No anomalies so Pam is assigned to task t_2^1 and Robert to task t_2^1.

Step 4: Run-time conditions necessitate updating CB based on constraint 5:
7. cannot_do_R_{Robert}(t_2^k) ←
   input(W_k).customer = “StarCo”
Since a new rule was added, we go back to step 2. Nothing changes so wait for new task instance.
An Example

Updated CB

1. related(Sam, Paul) ← Sam and Paul have a relationship.
2. related(Pam, Robert) ← Pam and Robert have a relationship.
3. cannot_do(u_i, t_{2}^k) ← ∀i, count_{nv}(u_i, executed_{u}(u_i,t_{2}^k), n), n > 2
   A user can only execute two instances of task t_{2}.
4. cannot_do(u_i, t_{2}^k) ← executed_{u}(u_i,t_{1}^i), n), input(W_k).customer = input(W_i).customer, related(u_i, u_j)
   A user cannot execute task t_{2} when the someone who has a relationship with them has performed task t_{1} in any other instance for the same customer.
5. must_execute_R(R_B, t_{2}^k) ← input(W_k).customer = "StarCo"
   Someone from Role R_B must perform task t_{2} when the customer is StarCo.
6. related(Sam, Pam) ←
7. cannot_do(Robert, t_{2}^k) ← input(W_k).customer = "StarCo"

Step 3: Workflow instance W_2 begun for customer StarCo.

Permitted_users^R(t_{2}^2) = \{Jane\},
Permitted_roles^R(t_{2}^2) = \{R_B\},
Denied_users^R(t_{2}^2) = \{Robert\},
Denied_roles^R(t_{2}^2) = \emptyset

No anomalies so Paul is assigned to task t_{1}^2 and Jane is assigned to task t_{2}.

Step 4: Run-time conditions necessitate updating CB based on constraint 5:

8. cannot_do(Jane, t_{2}^k) ← input(W_k).customer = "StarCo"

Since a new rule was added, we go back to step 2.
**An Example**

**Updated CB**

1. related(Sam, Paul) ← Sam and Paul have a relationship.
2. related(Pam, Robert) ← Pam and Robert have a relationship.
3. cannot_do(u, t2k) ← \( \forall i, \) count_nv(u, executed, \( u, t2i), n), \( n > 2 \)
   A user can only execute two instances of task \( t2 \).
4. cannot_do(u, t2k) ← executed(u, t1i), n), input(W_k).customer = input(W_i).customer, related(u, u1)
   A user cannot execute task \( t2 \) when the someone who has a relationship with them has performed task \( t1 \) in any other instance for the same customer.
5. must_execute_R(R_B, t2k) ← input(W_k).customer = "StarCo"
   Someone from Role R_B must perform task \( t2 \) when the customer is StarCo.
6. related(Sam, Pam) ←
7. cannot_do(Robert, t2k) ← input(W_k).customer = "StarCo"
8. cannot_do(Jane, t2k) ← input(W_k).customer = "StarCo"

---

**Step 3: Workflow instance \( W_3 \) begun for customer StarCo.**

**Permitted_users**\( ^R(t2^3) = \emptyset \)**
**Permitted_roles**\( ^R(t2^3) = \{ R_B \} \)**
**Denied_users**\( ^R(t2^3) = \{ Robert, Jane \} \)**
**Denied_roles**\( ^R(t2^3) = \emptyset \)

**Depletion anomaly – no one to assign to task \( t2^3 \). System specific handling would take place.**
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. **Delegation of authority**
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
5. Delegation of Authority

- Delegation is an important concept in workflow systems to ensure that work can be completed even if the users/roles to perform a specific task are not available.
  - tasks to users
  - roles to users
  - roles to roles

- Delegator ➔ Delegatee

Ex: Manager ➔ Subordinate
5. Delegation of Authority

- Delegations ↔ Workflow authorization constraints
- Consistency checking and task assignment:
  - 1\textsuperscript{st} static check: there is no delegation cycle.
  - 2\textsuperscript{nd} static check: the delegation is not inconsistent with authorization constraints.
    - No user obliged to perform a task should be allowed to delegate.
    - No user who is restricted from doing a task should be the recipient of a delegation of that task.
  - At run-time: delegations are further evaluated to make actual assignments when the delegation is of a role or to a role
5. Delegation of Authority

- Some conditions of delegation:

<table>
<thead>
<tr>
<th>Delegation Conditions</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal Delegation</td>
<td>They allow a user to constraint delegation of a task to a defined time interval</td>
</tr>
<tr>
<td>Workload Delegation</td>
<td>They allow a user to constrain delegation of a task to a workload level.</td>
</tr>
<tr>
<td>Value Delegation</td>
<td>They allow a user to constrain delegation of a task depending upon attributes of the task</td>
</tr>
</tbody>
</table>
5. Delegation of Authority

- The revoking of delegation in the middle of the workflow when the conditions no longer apply or the delegation is revoked:
  - The delegatee has not begun the task and is not yet assigned.
  - The task is already assigned but not completed.
  - The task is begun.

- A delegatee should never be allowed to perform a task that the delegator is not authorized to perform.
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. **Conflict of interest**
7. Workflow safety analysis
8. Open issues
6 Conflict-of-Interest

- It is a well know concept.
- A situation in which sb has two jobs, aims, roles, etc. and cannot treat both of them equally and fairly at the same time (Oxford Advanced Learner's Dictionary - 8th Edition)

- Execution of inter-organization workflows may raise a number of security issues including conflict of interest among competing organizations.
  - Mobile software agents.
  - Decentralized environment.
6 Conflict-of-Interest

- Brewer and Nash model: Chinese wall model:
  - User are allowed access to information not in conflict with any other information that they already possess.
  - The company information is categorized into mutually disjoint conflict of interest classes.

Figure 5-15  The Chinese Wall model provides dynamic access controls.
6 Conflict-of-Interest
A Chinese Wall Security Model for Decentralized Workflow Systems

- The model uses the notion of self-describing workflows and WFMS stubs.
- Self-describing workflows are partitions of a workflow that carry sufficient information so that they can be managed by a local task execution agent rather than a central WFMS.
- A WFMS stub is a light-weight component that can be attached to a task execution agent, which is responsible for receiving the self-describing workflow, modifying it and re-sending it to the next task execution.
Web Services are now being touted as the way to coordinate business entities in an inter-organizational business process.

Research on determining security policies that should apply to the overall process is even further behind.

The organizations involved in the workflow may be unfamiliar to each other.

Languages and mechanisms for exchanging policy information
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. Conflict of interest
7. **Workflow safety analysis**
8. Open issues
7. Workflow Safety Analysis

- Safety analysis
- The safety problem: “Is there a reachable state in which a particular subject possesses a particular privilege for a specific object?

  (first identified by Harrison, Ruzzo and Ullman)

- Analyzing the safety of the WAM: colored Petri nets (Atluri and Huang)
  - Petri nets: provide a combination of specification and modeling tools to depict the system behavior and formal verification tools
7. Workflow Safety Analysis (cont)

- Petri nets:
  - allows a smooth transition from the conceptual level to an implementation of a workflow
  - have the advantage of visually depicting properties, relationships and restrictions among tasks of a given work.
  - helps one to understand the implications of the authorization policies
Safety analysis: especially important when task authorizations may be delegated and revoked as discussed (Schaad et al)

=> a model-checking based approach for automated analysis of delegation and revocation functionalities in the context of a workflow requiring static and dynamic separation of duty properties

The analysis on the state machine will determine whether a set of delegations and/or revocations may be safely accepted.
A CTPN representation of WAM for a Workflow with one task
A CTPN representation of the check processing example
1. Introduction
2. Security requirements in workflow systems
3. Workflow authorization model
4. Separation of duty
5. Delegation of authority
6. Conflict of interest
7. Workflow safety analysis
8. Open issues
8. Open Issues

- The main research: Interoperability and integration.
  - WMS integration with identity management systems, applications, and ERP systems: rather ad-hoc.
  - Interactions between security constraints and business rules need to be considered when user-task assignments are performed.
  - Control of authorizations and permissions can be set across the organization => the WMS only need to be concerned with constraints apply specifically to the workflow and could consult the policy management system for general authorization questions.
8. Open Issues (cont)

- Inter-organizational workflows are cobbled together from separate, disparate business processes within each organization => using the XML-based ebXML business process interface for support of transactions between organizations

- Security perspective: constraints can currently only be applied at the organizational level. No standard way exists for external organizations to specify constraints on assignment of individuals, adhering to the individual organization’s business rules and security constraints is essential while composing inter-organizational workflows
8. Open Issues (cont)

- Participating organizations need also to be concerned with evaluating the risk of working with other participating organizations contractual obligations must be established, monitored and assessed and audit trails must be available to all participants. Secure, available and reliable information on business process execution has not been deeply addressed in terms of inter-organizational business processes or workflows.

- Many of the research ideas presented have not been implemented and the problems they solve are still not addressed in existing systems.
Conclusion
Most commercial workflow systems provide minimal security features such as user authentication.

- Some commercial WFMSs such as FlowMark, Lotus Notes and Changengine can support role-based access control, they do not provide support to specify and enforce separation of duties constraints => implemented in an ad-hoc manner through a script type language (ad-hoc implementation makes specification, analysis and maintenance of security policies more difficult.)

- Many efforts in WFMS implementation have been placed in protecting data transmitted over the network, but little emphasis has been given on providing access control for workflow activities.
Conclusion (cont)

- The security requirements of workflow systems and discuss authorization, separation of duties, authentication and anonymity at length.
  - The need for synchronization of authorization flow with the workflow
  - A full fledged authorization system should consider the following additional requirements:
    - Assigning different roles to tasks based on the outcome of the prior task,
    - Granting different permissions to roles based on the outcome of the task
Conclusion (cont)

- Capability to specify different authorizations for different instances of the same workflow,
- Ability to specify authorizations based on the context and based on the responsibilities to be performed by individuals,
- Delegating the responsibility to other users and roles.

- Security in workflow systems is an active area of research, and many researchers around the world are investigating the above issues
Reference

- Vijayalakshmi Atluri and Janice Warner, Security for Workflow Systems,
- Vijayalakshmi Atluri and Wei-Kuang Huang, A Petri Net Based Safety Analysis of Workflow Authorization Models
- ....