2005

A Logic Based Modeling Approach to Managing Workflow Policy Changes

Harry J. Wang
University of Arizona, jiannan@eller.arizona.edu

J. Leon Zhao
University of Arizona, lzhao@eller.arizona.edu

A Logic-based Modeling Approach to Managing Workflow Policy Changes

Harry J. Wang
Department of MIS
University of Arizona
Tucson, AZ 85721
jiannan@eller.arizona.edu

J. Leon Zhao
Department of MIS
University of Arizona
Tucson, AZ 85721
lzhao@eller.arizona.edu

ABSTRACT

Workflow management systems are becoming increasingly important in the automation of business processes. In order to ensure proper workflow execution, workflow policies must be specified with respect to users, roles, and tasks. In today's dynamic business environment, successful organizations must be able to respond to new customer demands and market opportunities with flexibility and speed. However, without systematic management of workflow policies, changes in organizational structure and process models can lead to inconsistent workflow specifications. Thus far, research in the change management of workflow policies has been scant. In this paper, we propose a logic-based approach to address this problem. Our contribution is three-fold: 1) a modeling language based on predicate logic is proposed, which is succinct and expressive enough to represent process model, organization model, and workflow polices; 2) workflow policy consistency in a dynamic changing environment is formally defined and analyzed based on the proposed language. 3) two algorithms are developed to check and enforce the policy consistency. To the best of our knowledge, this is the first work focuses on the formal analysis of workflow policy change management.

Keywords

Workflow Management, Workflow Policy, Change Management, Predicate Logic, Modeling Language.

INTRODUCTION

Today, Workflow Management Systems (WFMSs) are widely used by organizations to coordinate activities, streamline business processes, and support e-business (Stohr and Zhao, 2001). A workflow usually consists of a set of well-defined tasks, which must be matched with agents (either human or machine) in accordance with workflow policies, e.g., a person cannot approve a check issued by him/herself (Bertino, et al., 1999; Georgakopoulos, et al., 1995). The workflow system must ensure that the workflow policies are consistent. However, given that large WFMSs are often deployed in organizations where the organizational model, the process model, the users, and roles change often, maintaining the consistency of workflow policies can be extremely difficult (Ribeiro and Guedes, 1999; van der Aalst, 2001). Moreover, in the era of on-demand business, companies need to change their business model frequently to respond to new customer demand and market opportunities. Dynamic changes in process and organization model can lead to inconsistent workflow policies without proper management, which may result in costly processes and system breaches. Although there has been a noticeable amount of research in adaptive workflow (Ellis and Keddara, 2000; Ellis, et al., 1995; van der Aalst, 2001) and workflow authorization constraints (Atluri and Huang, 1996; Bertino, et al., 1999; Casati, et al., 2001; Huang and Atluri, 1998; Wu, et al., 2002), there is lack of methodology for managing workflow policies under consistent process and organization changes.

In this paper, we propose a logic-based approach to workflow policy management. In particular, we present a modeling language based on predicate logic, which is used to represent, the process model, the organizational model, and workflow policies. Using this language, we formally define and analyze the consistency of workflow policies in a dynamic environment. Moreover, we develop algorithms to automatically check and enforce the consistency of workflow policies. System architecture and implementation issues are also discussed to integrate our approach into existing workflow management systems.

The rest of the paper is organized as follows. In the next section, we briefly review the literature in workflow authorization constraints, dynamic workflow changes, and process and organization modeling. Then we present a logic-based modeling language followed by the definition and analysis of workflow policy consistency in a dynamic environment. Subsequently, we present the algorithms for automatic checking and enforcement of workflow policy consistency and demonstrate their functionality via an example. Finally, we discuss architectural and implementation issues before concluding our research and outlining a future research plan.

BRIEF LITERATURE REVIEW

Workflow consists of a set of well-defined tasks. To ensure that these tasks are executed by appropriate personnel, certain authorization mechanisms must be in place. Role-based, task-based, temporal access control models have been developed to facilitate workflow authorization management (Wu, et al., 2002). Atluri and Huang (1996) proposed a workflow authorization model to synchronize authorization flow with the workflow. Bertino et al. (1999) presented a logic-based framework for the specification and enforcement of workflow authorization constraints. They categorize workflow security policies into three categories, namely, static, dynamic, and hybrid policies, where static policies can be evaluated without executing the workflow, dynamic policies can only be evaluated during the execution of workflow, and hybrid policies can by partially verified without executing the workflow. Their work focused on the automatic role and user assignment to tasks without violating any constraints. Casati et al. (2001) studied workflow authorization constraints using active database technology. A constraint is described by an ECA (event-condition-action) rule based on workflow instance, time, and history. However, none of those research efforts has thus far addressed the dynamic change problem of workflow policies.

The ability to respond effectively to changes is an important issue in WFMSs (van der Aalst, 2001). Workflow changes can cause inefficiencies, inconsistencies, and high costs if not properly handled (Ellis, et al., 1995). Different type of process changes have been discussed (Sadiq, et al., 2000) and mathematical formalisms have been proposed to analyze the process changes (Ellis, et al., 1995; van der Aalst, 2001). Until now, how process changes can affect workflow policies has not been explicitly studied.

The design of WFMSs requires the specification of process structures and organizational model. There has been extensive research efforts on the process modeling, resulting in many process modeling methods, such as UML activity diagram (OMG, 2003), Petri Nets (van der Aalst, 1998), and Logic-based approach (Bi and Zhao, 2004; Davulcu, et al., 1998). Besides the process flow perspective, organizational perspective is also important for workflow management systems (zur Muhlen, 2004). An organizational model is essential for the workflow engine to determine who must perform a task and what policies should be enforced (van der Aalst, et al., 2003). An organizational model represents the structure of an organization, which is usually a hierarchy of roles and users (Basu and Kumar, 2002; Stohr and Zhao, 2001). A user can play multiple roles and a role can have multiple users (Sandhu, et al., 1996). Several organizational meta models have been proposed to make WFMSs more "organizationally aware" (Cheng, 1999; Jablonski and Bussler, 1996; van der Aalst, et al., 2003; zur Muhlen, 2004). However, there is lack of a unified approach to modeling process, organization, and policies, making the analysis of workflow policies difficult. Our previous work (Wang and Zhao, 2004) studied the problem of workflow policy changes at a conceptual level; in this paper we extend our previous work by providing formal definitions and a logic-based analytical framework.

A WORKFLOW EXAMPLE

In this section, we present a workflow example that we will use throughout the rest of the paper. A requisition approval workflow is represented using an UML activity diagram in Figure1, and each task is described in Table 1. Note that t_6 , t_9 , and t_{13} ,which are underlined in the figure, are executed by machine agents,.

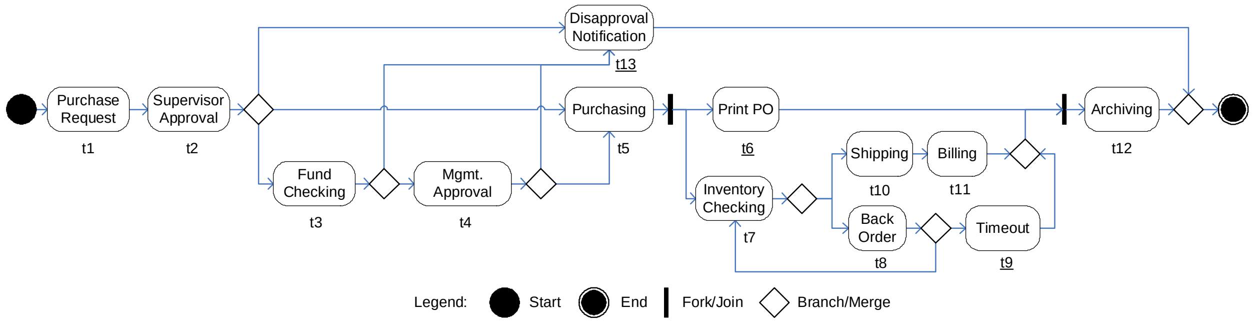

Figure 1. Requisition Approval Workflow

Task	***Name***	***Description***
t_1	Purchase Request	Employees submit purchase request.
t_2	Supervisor Approval	The supervisor of task initiator approves the request
t_3	Fund Checking	When request amount is greater than 1000, a checking is conducted to make sure there is sufficient fund.
t_4	Mgmt. Approval	When fund is sufficient, higher level manager needs to approve.
t_5	Purchasing	Purchase Request is reviewed and sent to inventory. PO is requested to be printed.
t_6	Print PO	PO is printed.
t_7	Inventory Checking	Inventory checking. If in stock, start shipping; otherwise back order.
t_8	Back Order	The requested item is back ordered.
t_9	Timeout	Timeout is trigger when back order cannot complete after a certain period of time. A timeout notification is also sent to the purchase requestor.
t_{10}	Shipping	Item is shipped.
t_{11}	Billing	Proper department is billed for the items ordered.
t_{12}	Archiving	The purchase transaction information is summarized and archived.
t_{13}	Disapproval Notification	The requestor is notified that his/her purchase request is declined via email.

Table1. Task Descriptions

The role hierarchy associated with the workflow is presented in Figure 2. The employee appointment information (user-role assignment) is as follows: GM: {John}; ED: {Joe}; PD: {Jason}; AD: {Maggie}; SE:{Eric, Ray}; PC:{Peter}; IC:{Sam}; DC:{Dan, Jack}; AC:{Steve, Ben}.

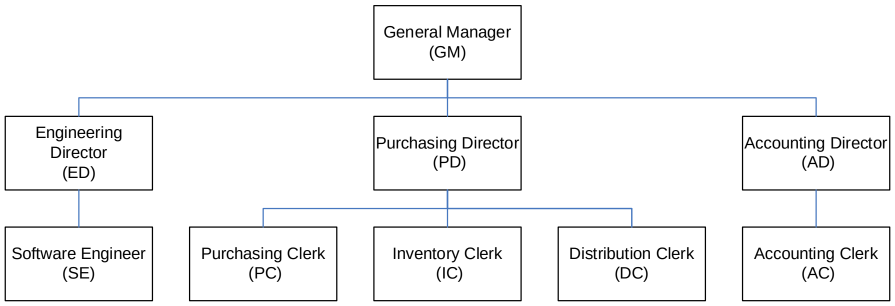

Figure 2: Organizational Structure

To ensure the proper execution of the workflow, roles and users may be assigned to tasks and workflow polices should be specified to enforce business requirements. The workflow policies for the requisition approval workflow are given below:

p_1: any employee can submit purchase request.

p_2: t_2 (Supervisor Approval) must be executed by the supervisor of the purchase requestor.

p_3: t_3 (Fund Checking) must be executed by an Accounting Clerk (AC).

p_4: the role associated with t_4 (Mgmt. Approval) must be the supervisor of the role that executes t_2.

p_5: t_5 (Purchasing) is handled by a Purchasing Clerk (PC)

p_6: t_7 (Inventory Checking) is conducted by an Inventory Clerk (IC).

p_7: t_{10} (Shipping) must be handled by a Distributed Clerk (DC).

p_8: t_{11} (Billing) must be handled by a Distributed Clerk (DC).

p_9: t_{12} (Archiving) is handled by a Purchasing Clerk (PC).

p_{10}: t_1 (Purchase Request) and t_3 (Fund Checking) cannot be done by the same person.

p_{11}: t_{10} (Shipping) and t_{11} (Billing) must be handled by the same person.

$p_{12} : t_6, t_9, t_{13}$ are executed by machine agents.

Policies p_1 through p_9 are role-task assignment policies. Policy p_{10} is an instance of separation of duties policy, whereas p_{11} is a binding of duties policy. Moreover, p_2, p_4, p_{11}, p_{12} are dynamic policies, which can be evaluated only during workflow execution. From this example we can see that the specification of workflow polices refers to the process model and organizational model for the information about tasks, users and roles. In the next section, we present a logic-based modeling language to unify the representation of the process, organization, and workflow policies.

A FORMAL LOGIC-BASED MODELING LANGUAGE

The modeling language is based on predicate logic and its syntax is similar to Prolog, one of the well-known logic programming languages (Rowe, 1988). Our language can be compiled and interpreted by most Prolog-based reasoning systems. In this language, facts are treated as explicit rules as their bodies are empty (Bertino, et al., 1999). There are four major components, namely, Process Rules, Organizational Rules, Policy Rules, and Auxiliary Rules.

A process model can be constructed from five primitive workflow blocks (WfMC, 1999) as shown in Figure 3.

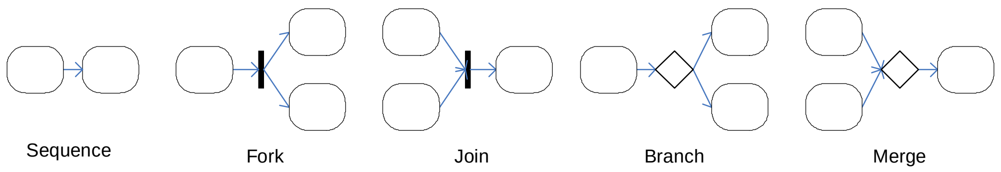

Figure 3. Building Blocks for Workflow Modeling

We use $T = \{t_i \mid i \in [1,...,n]\}$ to denote the set of tasks in a given workflow. t_s and t_e represent starting and ending nodes respectively. The set of predicates used to represent the five workflow building blocks are shown in Table 2.

Predicate	Description
$next(t_i, t_j)$	task t_i directly leads to task t_j
$fork(t_i, [t_{j_1}, t_{j_2}, ..., t_{j_n}])$	task t_i leads to a list of parallel tasks $[t_{j_1}, t_{j_2}, ..., t_{j_n}]$
$join([t_{i_1}, t_{i_2}, ..., t_{i_n}], t_j)$	a list of parallel tasks $[t_{i_1}, t_{i_2}, ..., t_{i_n}]$ converges to task t_j
$branch(t_i, [t_{j_1}, t_{j_2}, ..., t_{j_n}])$	after task t_i exactly one of the tasks in $[t_{j_1}, t_{j_2}, ..., t_{j_n}]$ is to be executed
$merge([t_{i_1}, t_{i_2}, ..., t_{i_n}], t_j)$	any task in $[t_{i_1}, t_{i_2}, ..., t_{i_n}]$ can trigger task t_j

Table 2. Process Modeling Predicates

Using these predicates, the requisition approval workflow in Figure 1 can be expressed as shown in Figure 4. Note that whenever Fork/Join and Branch/Merge are directly connected in a process model, a dummy task t_d is inserted in between to help the modeling.

$next(t_s, t_1)$, $next(t_1, t_2)$, $branch(t_2, [t_3, t_5, t_{13}])$, $branch(t_3, [t_4, t_{13}])$, $branch(t_4, [t_5, t_{13}])$, $fork(t_5, [t_6, t_7])$, $branch(t_7, [t_8, t_{10}])$, $next(t_{10}, t_{11})$, $branch(t_8, [t_7, t_9])$, $merge([t_9, t_{11}], t_{d_1})$, $join([t_6, t_{d_1}], t_{12})$, $merge([t_{12}, t_{13}], t_e)$

Figure 4. Logic Representation of the Workflow in Figure 1

In order to help with process analysis, we define the concept of *Task Reachability*. We say that t_j is reachable from t_i, if there exists at least one path from t_i to t_j in the process model. Intuitively, one task can be reached by another task if they are next to each other or they are directly connected to a routing construct that is of the type Fork, Join, Branch, or Merge. Task reachability is recursively defined in Figure 5. $member(t_i, [t_{i_1}, t_{i_2}, .., t_{i_n}])$ means that t_i is a member of the list $[t_{i_1}, t_{i_2}, .., t_{i_n}]$.

Definition 1: Task Reachability

$reach(t_i,t_j) \leftarrow next(t_i,t_j)$
$reach(t_i,t_j) \leftarrow fork(t_i,[t_{j_1},t_{j_2},..,t_{j_n}]), member(t_j,[t_{j_1},t_{j_2},..,t_{j_n}])$
$reach(t_i,t_j) \leftarrow join([t_{i_1},t_{i_2},..,t_{i_n}],t_j), member(t_i,[t_{i_1},t_{i_2},..,t_{i_n}])$
$reach(t_i,t_j) \leftarrow branch(t_i,[t_{j_1},t_{j_2},..,t_{j_n}]), member(t_j,[t_{j_1},t_{j_2},..,t_{j_n}])$
$reach(t_i,t_j) \leftarrow merge([t_{i_1},t_{i_2},..,t_{i_n}],t_j), member(t_i,[t_{i_1},t_{i_2},..,t_{i_n}])$
$reach(t_i,t_j) \leftarrow next(t_i,t_k), reach(t_k,t_j)$
$reach(t_i,t_j) \leftarrow reach(t_j,t_i)$

Figure 5. Definition of Task Reachability

Now, we can use queries to analyze some properties of the workflow. For instance, query $?- reach(t_9,t_{11})$ returns No, because t_9 and t_{11} are on the different branches of a Branch construct. Moreover, we assume that if a task t_i exists in the workflow $?- reach(t_s,t_i)$ is always true.

In order to model organizational model, we propose a set of organization predicates and rules as shown in Table 3. We use $R = \{r_i \mid i \in [1,...,m]\}$ and $U = \{u_i \mid i \in [1,...,l,ma]\}$ to denote a set of roles and a set of users respectively. In particular, u_{ma} is used to represent a machine agent. The role privilege inheritance rule says that a role inherits all the privileges of the roles that it manages. The user privilege inheritance rule is defined similarly.

Organization Rules	Description
$manage(r_i,r_j)$	r_i manages r_j
$manage(u_i,u_j)$	For all $u_i \in r_i, u_j \in r_j$ if r_i manages r_j then u_i manages u_j
$role(u_i,r_j)$	User u_i plays role r_j
$manage(r_i,r_j) \leftarrow manage(r_i,r_k), manage(r_k,r_j)$	role privilege inheritance rule:
$manage(u_i,u_j) \leftarrow manage(u_i,u_k), manage(u_k,u_j)$	user privilege inheritance rule:

Table 3. Organizational Rules

The organizational model in Figure 2 can be expressed as the logical statements in Figure 6. Note that because GM is the highest manager, he/she manages him/herself denoted by $manage(GM,GM)$. We also assume there is always a role named "Machine Agent (MA)" with users represented by u_{ma}.

manage(GM,GM), manage(GM,ED), manage(GM,PD), manage(GM,AD), manage(ED,SE), manage(PD,PC), manage(PD,IC), manage(PD,DC), manage(AD,AC), role(John,GM), role(Joe,ED), role(Jason,PD), role(Maggie,AD), role(Eric,SE), role(Ray,SE), role(Peter,PC), role(Sam,IC), role(Jack, DC), role(Dan, DC), role(Steve,AC), role(Ben, AC), role(u_{ma} *,MA)*

Figure 6. Logical Representation of the Organizational Model in Figure 2

Now we can reason about the organizational model. For instance, the answer to the query $?- manage(X,Eric)$ shows the supervisor of Eric, which is Joe in this case. Query $?- manage(Jason,Eric)$ returns No, because Jason is not the manager of Eric.

One important step in workflow design is to assign resources to tasks and specify policies to constraint task assignments and execution. To facilitate this design step, we propose task assignment rules as shown in Table 4.

Task Assignment Rules	Description
$r\_execute(r_i,t_j)$	Role r_i is assigned to execute task t_j
$cannot\_r\_execute(r_i,t_j)$	Role r_i is not allowed to execute task t_j
$u\_execute(u_i,t_j)$	User u_i is assigned to execute task t_j

$must\_u\_execute(u_i, t_j)$	User u_i must execute task t_j
$cannot\_u\_execute(u_i, t_j)$	User u_i is not allowed to execute task t_j
$u\_execute(u_i, t_j) \leftarrow must\_u\_execute(u_i, t_j)$	u_i must execute task t_j implies u_i execute task t_j
$u\_execute(u_i, t_j) \leftarrow role(u_i, r_k), r\_execute(r_k, t_j)$	If r_k can execute task t_j and u_i belongs to r_k then u_i can execute task t_j
$cannot\_u\_execute(u_i, t_j) \leftarrow role(u_i, r_k), cannot\_r\_execute(r_k, t_j)$	If r_k cannot execute task t_j and u_i belongs to r_k then u_i cannot execute task t_j

Table 4. Task Assignment Rules

Then the workflow policies defined for the requisition approval workflow can be formally expressed as follows:

p_1: $r\_execute(r_i, t_1)$
p_2: $r\_execute(r_i, t_2) \leftarrow manage(r_i, r_k), r\_execute(r_k, t_1)$
p_3: $r\_execute(AC, t_3)$
p_4: $r\_execute(r_i, t_4) \leftarrow manage(r_i, r_k), r\_execute(r_k, t_2)$
p_5: $r\_execute(PC, t_5)$
p_6: $r\_execute(IC, t_7)$
p_7: $r\_execute(DC, t_{10})$
p_8: $r\_execute(DC, t_{11})$
p_9: $r\_execute(PC, t_{12})$
p_{10}: $cannot\_u\_execute(u_i, t_3) \leftarrow u\_execute(u_i, t_1)$ or $cannot\_u\_execute(u_i, t_1) \leftarrow u\_execute(u_i, t_3)$
p_{11}: $must\_u\_execute(u_i, t_{11}) \leftarrow u\_execute(u_i, t_{10})$ or $must\_u\_execute(u_i, t_{10}) \leftarrow u\_execute(u_i, t_{11})$
p_{12}: $u\_execute(u_{ma}, t_6)$, $u\_execute(u_{ma}, t_9)$, $u\_execute(u_{ma}, t_{13})$

Figure 7. Logical Representation of the Workflow Polices

In order to facilitate the formal analysis, some auxiliary predicates are necessary, which are presented in Table 5.

Auxiliary Rules	Description
$count(Q, n)$	Count the number of different answers of query Q and return the value to n
$findall(Q, S)$	Find all the different answers of query Q and return them to a set S
$findone(Q, S)$	Find only one answer of query Q if any and return it to a set S
$=, <, >, \leq, \geq$	Comparison Predicates

Table 5. Auxiliary Rules

Some useful queries are explained as follows:

$count(u\_execute(u_i, t_3), n)$: this returns the total number of users assigned to task t_3 (Fund Checking), which is 2.
$findall(reach(t_s, t_i), S_t)$: this returns all the tasks in the workflow. For our example, $S_t = \{t_i \mid i \in [1, ..., 11]\}$
$manage(r_i, r_i)$, $findall(manage(r_i, r_j), S_r)$: this returns all the roles in the organization model. For our example, S_r={GM, ED, PD, AD, SE, PC, IC, DC, AC,MA}
$findall(role(u_i, r_j), S_{r+u})$: this returns all the roles and users in the organization. The set of users can be retrieved by $S_u = S_{r+u} \setminus S_r$.

The union of process rules R_s in Figure 4, organization rules R_o in Figure 6, and policy rules R_p in Figure 7 is referred as ***Workflow Specification*** (WS). In fact, the goal of conceptual workflow design is to produce a consistent WS such that the workflow can be executed successfully.

Definition 2: Workflow Specification (WS) Consistency

$findall(reach(t_s,t_i),S_t)$, $findall(role(u_i,r_j),S_{r+u})$, $manage(r_i,r_i)$, $findall(manage(r_i,r_j),S_r)$, $S_u = S_{r+u} \setminus S_r$

A WS is consistent if the following conditions hold:

1. $\forall t_i \in S_t, \exists u_j \in S_u$ such that $count(u\_execute(u_j,t_i),n), n \geq 1$
2. $\nexists u_i \in S_u$, such that $u\_execute(u_i,t_j) \in WS$ and $cannot\_u\_execute(u_i,t_j) \in WS$

   $\nexists r_i \in S_r$, such that $r\_execute(r_i,t_j) \in WS$ and $cannot\_r\_execute(r_i,t_j) \in WS$

Figure 8. Definition of Workflow Specification Consistency

We define that a *WS* is consistent if 1) at least one user has been assigned to each task; 2) there is no contradictory information in *WS*. For instance, $u\_execute(u_1,t_1)$ and $cannot\_u\_execute(u_1,t_1)$ are contradictory due to the fact that they cannot be satisfied at the same time. The definition of *WS* consistency is formalized as shown in Figure 11.

ANALYSIS OF WORKFLOW POLICY CHANGES

Business processes must change with the customer demands and market opportunities. Due to acquisitions and mergers, companies' organization models also often undertake changes. Without a systematic management, these changes can lead to inconsistent workflow policies. In this section, we apply the logic-based language proposed in the previous section to analyze the changes of workflow policies.

Process model changes can be categorized into three categories: 1) task insertion, 2) task deletion, 3) structure modification (Sadiq, et al., 2000). Similarly, there are three types of organization model changes, namely, 1) insertion and deletion of roles 2) insertion and deletion of users 3) relocation of roles and users. The changes in process model and organizational model correspond to the changes in process rules R_s and organization rules R_o, resulting in a new workflow specification *WS*. Now the question is whether the new *WS* is consistent or not. In order to cope with the changes, the previous definition of *WS* consistency needs to be extended. Intuitively, the existence of the users, roles and tasks of the logic statements in *WS* must be verified. The extended *WS* consistency is formally defined as follows:

Definition 3: Extended Workflow Specification (*WS*) Consistency

$findall(reach(t_s,t_i),S_t)$, $findall(role(u_i,r_j),S_{r+u})$, $manage(r_i,r_i)$, $findall(manage(r_i,r_j),S_r)$, $S_u = S_{r+u} \setminus S_r$

A WS is consistent if the following conditions hold:

1. $\forall t_i \in S_t, \exists u_j \in S_u$ such that $count(u\_execute(u_j,t_i),n), n \geq 1$
2. $\nexists u_i \in S_u$, such that $u\_execute(u_i,t_j) \in WS$ and $cannot\_u\_execute(u_i,t_j) \in WS$

   $\nexists r_i \in S_r$, such that $r\_execute(r_i,t_j) \in WS$ and $cannot\_r\_execute(r_i,t_j) \in WS$
3. $S = S_t \cup S_r \cup S_u$

   $findall(r\_execute(r_i,t_j),S_1)$ such that $S_1 \subseteq S$

   $findall(cannot\_r\_execute(r_i,t_j),S_2)$ such that $S_2 \subseteq S$

   $findall(u\_execute(u_i,t_j),S_3)$ such that $S_3 \subseteq S$

   $findall(cannot\_u\_execute(u_i,t_j),S_4)$ such that $S_4 \subseteq S$

   $findall(must\_u\_execute(u_i,t_j),S_5)$ such that $S_5 \subseteq S$

Figure 9. Definition of Extended Workflow Specification Consistency

Workflow Specification Consistency Checking and Enforcement

Based on the new definition of workflow specification consistency, we develop an algorithm for the automatic consistency checking as shown in Figure 10.

```
Algorithm 1: Workflow Specification (WS) Consistency Checking Algorithm
INPUT: 1) Workflow Specification including process rules $R_s$, organization rules $R_o$, and policy rules $R_p$
OUTPUT: 1) SUCCESS if the $WS$ is consistent
        2) REDUNDANT if there is at least one redundant policy
        3) INVALIDATED if there is at least one invalidated policy
        4) CONTRADICT if there is at least one pair of contradict policies
        5) MISSING if there is at least one task with no user assigned to it

  $findall(reach(t_s, t_i), S_t)$
  $findall(role(u_i, r_j), S_{r+u})$
  for each logic statement $st_i$ in $R_p$ :
      execute: $findall(r\_execute(r_i, t_j), S_1)$, $findall(cannot\_r\_execute(r_i, t_j), S_2)$, $findall(u\_execute(u_i, t_j), S_3)$
       $findall(cannot\_u\_execute(u_i, t_j), S_4)$, $findall(must\_u\_execute(u_i, t_j), S_5)$
      compute $S = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5$
      if $S \subseteq S_t \cup S_{r+u}$
          break
          elseif $S \cap (S_t \cup S_{r+u}) = \varnothing$
              return REDUNDANT and $st_i$
              exit
                  elseif
                      return INVALIDATED, $st_i$ and $S \setminus (S \cap (S_t \cup S_{r+u}))$
                      exit
                  endif
          endif
      endif
  endfor
  execute $findone(\{r\_execute(r_i, t_j), cannot\_r\_execute(r_i, t_j)\}, S_6)$
          if $S_6 \neq \varnothing$
             return CONTRADICT and $S_6$
             exit
          endif
  execute $findone(\{u\_execute(u_i, t_j), cannot\_u\_execute(u_i, t_j)\}, S_7)$
          if $S_7 \neq \varnothing$
             return CONTRADICT and $S_7$
             exit
          endif
  for each $t_i \in S_t$
      execute $count(u\_execute(u_j, t_i), n)$
      if $n = 0$
         return MISSING and $t_i$
         exit
      endif
  endfor
  return SUCCESS
```

Figure 10. Workflow Specification Consistency Checking Algorithm

Given a WS, Algorithm 1 cannot only check its consistency, but also pinpoint each policy that causes the inconsistency if any.

In particular, the policies causing the inconsistency are classified into four categories, namely, ***Redundant***, ***Invalidated***, ***Contradict***, and ***Missing*** policies. Proper actions must be taken according to the type of the policy to remove the inconsistency. Redundant policies can just be deleted or disabled. Invalidated policies can be further subdivided into policies invalidated due to the deletion of tasks and policies invalidated due to the deletion of users and roles. The first type of invalidated policies can be removed without further investigations if all the tasks involved in the policy have been deleted, otherwise appropriate modifications to the policy are needed. For the second type of invalidated policies, the users and roles that have been deleted must be replaced properly, where privilege inheritance and delegation may provide solutions. For contradictory polices, further study of the policies is needed to solve the contradiction, which is out of the scope of this paper. For missing polices, appropriate role-task assignments can address the problem.

As we mentioned before, dynamic security policies, such as separation of duties and binding of duties can only be evaluated during the execution of workflow. In the workflow execution phase, when a task is activated, we use Algorithm 2 to enforce the WS consistency.

```
Algorithm 2. Workflow Specification (WS) Consistency Enforcement Algorithm
INPUT: 1) Workflow Specification checked using Algorithm 1.
OUTPUT: 1) CHECK and Updated Policy Rules $R_p$ if the algorithm completes successfully, otherwise
            2) FAIL and the task $t_i$ where at least one policy cannot be enforced.

upon each activation of task $t_i$ do:
   execute $count(must\_u\_execute(u_j, t_i), n)$ // enforce binding of duties
   if $n \neq 0$
      return CHECK and exit
   endif
   execute $count(u\_execute(u_j, t_i), n)$
   for $k = 1$ to $n$ // enforce separation of duties
      add $u\_execute(u_{j_k}, t_i)$ to $R_p$
      run Algorithm 1
      if Algorithm 1 returns SUCCESS
         return CHECK and exit
      else
         remove $u\_execute(u_{j_k}, t_i)$ from $R_p$
      endif
      $k = k + 1$
   endfor
   return FAIL and $t_i$
```

Figure 11. Workflow Specification Consistency Enforcement Algorithm

To illustrate the proposed algorithms, assume the requisition approval workflow changes to the one shown in Figure 12. In particular, task t_7 (Inventory Checking) and t_{11} (Billing) is merged into t_5 (Purchasing) as the result of process redesign. Due to the adoption of a new technology, all the transaction information can be archived automatically by the system, which causes the deletion of task t_{12} (Archiving). Furthermore, a new task t_{14} (Notify Purchase Requestor) is added to notify the purchase requestor that the item has been shipped or is out of stock. Again, task t_6, t_9, t_{13}, t_{14} as underlined, are executed by machine agents.

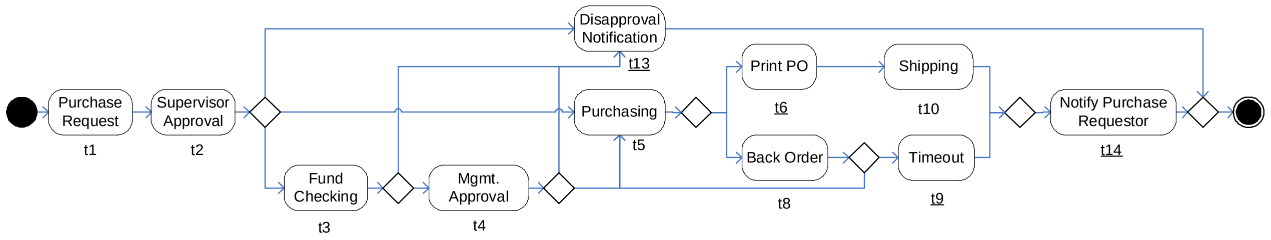

Figure 12. Modified Requisition Approval Workflow

The new process rules (R_s) are shown in Figure 13.

$next(t_s,t_1)$, $next(t_1,t_2)$, $branch(t_2,[t_3,t_5,t_{13}])$, $branch(t_3,[t_4,t_{13}])$, $branch(t_4,[t_5,t_{13}])$, $branch(t_5,[t_6,t_8])$, $branch(t_8,[t_5,t_9])$, $next(t_6,t_{10})$, $merge([t_9,t_{10}],t_{14})$, $merge([t_{13},t_{14}],t_e)$

Figure 13. New Process Rules

There are also some changes in the organizational model. In particular, role Inventory Clerk (IC) is removed and one accounting clerk Ben leaves the company. The new organization rules are shown in Figure 14.

manage(GM,GM), manage(GM,ED), manage(GM,PD), manage(GM,AD), manage(ED,SE), manage(PD,PC), manage(PD,DC), manage(AD,AC), manage(GM,MA), role(John,GM), role(Joe,ED), role(Jason,PD), role(Maggie,AD), role(Eric,SE), role(Ray,SE), role(Peter,PC), role(Dan, DC), role(Jack, DC), role(Steve,AC), role(u_{ma} *,MA)*

Figure 14. Modified Organization Rules

In order to check the consistency of the workflow policies after the changes in process and organization model take place. We can use the new Workflow Specification (WS) as the input to run Algorithm 1, whose result is shown in the following table.

#	Output	Description	Action(s)
1	REDUNDANT and $p_6 : r\_execute(IC,t_7)$	p_6 is redundant because of the deletion of IC and t_7	Delete p_6
2	INVALIDATED and $p_8 : r\_execute(DC,t_{11})$	p_8 is invalidated because of the deletion of t_{11}	Delete p_8
3	INVALIDATED and $p_9 : r\_execute(PC,t_{12})$	p_9 is invalidated because of the deletion of t_{12}	Delete p_9
4	INVALIDATED and p_{11}	p_{11} is invalidated because of the deletion of t_{12}	Delete p_{11}
5	MISSING and t_{14}	t_{14} is new and no user is assigned to it.	Assign a machine agent u_{ma} to t_{14}, Add $p_{13} : u\_execute(u_{nh},t_{14})$ to R_p
6	SUCCESS	WS is consistent	N/A

Table 6. Results of running Algorithm 1 on the Workflow Specification after Changes

The new policy rules includes $p_1, p_2, p_3, p_4, p_5, p_7, p_{10}, p_{12}, p_{13}$, which are consistent. During workflow execution phase, any employee can initiate a purchase request and Algorithm 2 is used to enforce the dynamic policies when each task activates. Suppose the accounting clerk Steve initiates a purchase request with an amount of $5000. Then $u\_execute(Steve,t_1)$ is inserted into R_p after t_1 completes. According to the policy p_2, accounting director Maggie will be assigned to execute t_2 and $u\_execute(Maggie,t_2)$ is added to R_p. Algorithm 2 returns CHECK for both t_1 and t_2. However, when task t_3 is activated, Algorithm 2 returns FAIL. This is because of the separation of duties policy p_{10}: t_1 (Purchase Request) and t_3 (Fund Checking) cannot be done by the same person. $u\_execute(Steve,t_1)$ implies $cannot\_u\_execute(Steve,t_3)$, so Algorithm 2 prevents Steve from executing t_3 and tries to find another eligible user for t_3. However, due to Ben's leave, Steve is the only user of role Accounting Clerk that is assigned to execute t_3 . In this case, Algorithm 2 fails and the WS consistency cannot be enforced. One possible solution is to assign Steve's manager Maggie to handle t_3 , but the detailed discussion on this issue is out of the scope of this paper.

The modeling language we proposed is not intended as the end-user language for modeling process, organization and workflow policies. It is an internal language used by system to conduct formal analysis. Next, we propose a system architecture for integrating our approach with exiting WFMSs and related implementation issues are also discussed.

System Architecture and Implementation Issues

In order to incorporate the change management functionality we propose into WFMSs, we develop a system architecture as shown in Figure 15. We assume that process model, organization model, and workflow policies are found in some existing WFMSs, which requires the mapping component to convert the definitions into our logic representation. When the automatic mapping is not available, a GUI tool is provided for the process designer to convert the proprietary definitions into our logic format. The algorithms we propose are implemented in the Policy Checking and Enforcement component, which provides APIs to workflow engines to do dynamic policy consistency analysis. Our approach uses a logic reasoning engine to do the analysis. Given that our language is very similar to Prolog, many free Prolog-based systems can be directly adopted, e.g., B-Prolog (http://www.probp.com/) and Strawberry Prolog (http://www.dobrev.com/).

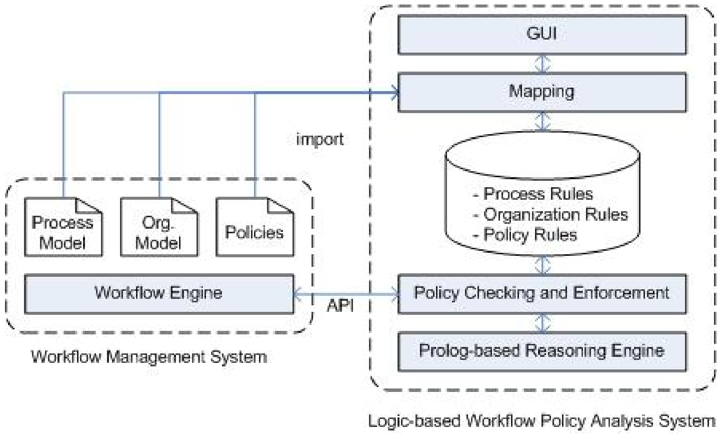
GUI
Mapping
import
- Process Rules
- Organization Rules
- Policy Rules
Process Model
Org. Model
Policies
Workflow Engine
API
Policy Checking and Enforcement
Prolog-based Reasoning Engine
Workflow Management System
Logic-based Workflow Policy Analysis System

Figure 15. System Architecture

CONCLUSION

In this paper, we proposed a logic-based approach to the analysis and enforcement of workflow policy consistency. Our contribution is threefold: First, we proposed a formal language based on predicate logic to unify the modeling of process, organization and workflow policies. We demonstrated that our language is expressive and can model all the basic workflow constructs, role hierarchy with privilege inheritance, and workflow policies, such as role-based and user-based task assignment, separation of duties, and binding of duties.

Second, based on the analysis of various changes in the process model and organization model, we formally defined a framework of workflow policy consistency in a dynamic environment using our language. To the best of our knowledge, this is the first attempt on the formal analysis of workflow policy changes.

Third, we developed algorithms to automatically check and enforce the consistency of workflow policies. The algorithms cannot only check whether the policies are consistent after changes in process and organization model take place, but also pinpoint the policies causing the inconsistency, which are categorized into four types: Redundant, Invalid, Contradict, and Missing policies. According to the type of the policy, proper actions to resolve the inconsistency are also explained. The functionality of the algorithms is demonstrated via an example. A system architecture is also presented to provide guidelines for integrating our approach into existing workflow management systems.

Our future research plan includes 1) enhancing the language to support more complex organization model such as teams and delegation; 2) enhancing the language to support more policies, such as task-based, temporal, team-based policies; 3) developing a proof-of-concept system to further validate the logic-based approach.

REFERENCES

1. Atluri, V., and Huang, W.-k. (1996) "An Authorization Model for Workflows," Proceedings of the Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security, pp. 44 - 64.
2. Basu, A., and Kumar, A. (2002) "Research commentary: Workflow management issues in e-Business," Info. Systems Res. 13, 1, pp. 1-14.
3. Bertino, E., Ferrari, E., and Atluri, V. (1999) "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," ACM Transactions on Information and System Security 2, 1, pp. 65-104.
4. Bi, H.H., and Zhao, J.L. (2004) "Applying propositional logic to workflow verification," Information Technology and Management 5, 3-4, pp. 293-318.
5. Casati, F., Castano, S., and Fugini, M. (2001) "Managing Workflow Authorization Constraints through Active Database Technology," Information Systems Frontiers 3, 3, pp. 319-338.
6. Cheng, E.C. (1999) "An Object-Oriented Organizational Model to Support Dynamic Role-based Access Control in Electronic Commerce Applications," Proceedings of the Proceedings of the 32nd Hawaii International Conference on System Sciences, pp. 8022.
7. Davulcu, H., Kifer, M., Ramakrishnan, C.R., and Ramakrishnan, I.V. (1998) "Logic based modeling and analysis of workflows," Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems pp. 25-33.
8. Ellis, C., and Keddara, K. (2000) "ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems," Computer Supported Cooperative Work 9, pp. 293-333.
9. Ellis, C., Keddara, K., and Rozenberg, G. (1995) "Dynamic change within workflow systems," Proceedings of the Proceedings of conference on Organizational computing systems, pp. 10 - 21.
10. Georgakopoulos, D., Hornick, M., and Sheth, A. (1995) "An overview of workflow management: From Process Modeling to Workflow Automation," Distributed and Parallel Databases 3, 2, pp. 119-153.
11. Huang, W.-K., and Atluri, V. (1998) "Analyzing the Safety of Workflow Authorization Models," Proceedings of the 12th IFIP Working Conference on Database Security,
12. Jablonski, S., and Bussler, C. (1996) Workflow Management: Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, London, UK.
13. OMG (2003) "Unified Modeling Language, Version 1.5, formal/03-03-01," Object Management Group, Mar. 2003.
14. Ribeiro, C., and Guedes, P. (1999) "Verifying workflow processes against organization security policies," Proceedings of IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '99) pp. 190-191.
15. Rowe, N.C. (1988) Artificial Intelligence Through Prolog, Prentice Hall,
16. Sadiq, S.W., Marjanovic, O., and Orlowska, M.E. (2000) "Managing change and time in dynamic workflow processes," International Journal of Cooperative Information Systems 9, 1-2, pp. 93-116.
17. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., and Youman, C.E. (1996) "Role-based Access Control Models," IEEE Computer 29, 2, pp. 38-47.
18. Stohr, E.A., and Zhao, J.L. (2001) "Workflow automation: overview and research issues," Info. Systems Frontiers: Special Issue on Workflow Automation 3, 3, pp. 281-296.
19. van der Aalst, W.M.P. (1998) "The application of Petri nets to workflow management," Journal of Circuits, Systems and Computers 8, 1, pp. 21-66.
20. van der Aalst, W.M.P. (2001) "Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change," Information Systems Frontiers 3, 3, pp. 297-317.
21. van der Aalst, W.M.P., Kumar, A., and Verbeek, H.M.W. (2003) "Organizational modeling in UML and XML in the context of workflow systems," Proceedings of the Proceedings of the 2003 ACM symposium on Applied computing, pp. 603 - 608.
22. Wang, H.J., and Zhao, J.L. (2004) "Change Management of Workflow Security Policies," Proceedings of the Pre-ICIS Workshop on Process Management and Information Systems, Washington D. C.,
23. WfMC (1999) "Workflow Management Coalition Terminology & Glossary, WfMC-TC-1011, Issue 3.0," Workflow Management Coalition, February 1999.
24. Wu, S., Sheth, A., Miller, J., and Luo, Z. (2002) "Authorization and Access Control of Application Data in Workflow Systems," Journal of Intelligent Information Systems 18, pp. 71-94.
25. zur Muhlen, M. (2004) "Organizational Management in Workflow Applications - Issues and Perspectives," Information Technology and Management 5, 3, pp. 271-291.