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*

Follow this and additional works at: [http://aisel.aisnet.org/amcis2005](http://aisel.aisnet.org/amcis2005)

**Recommended Citation**

[http://aisel.aisnet.org/amcis2005/320](http://aisel.aisnet.org/amcis2005/320)
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 policies; 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  

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 be 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 Figure 1, 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](image-url)
<table>
<thead>
<tr>
<th>Task</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>Purchase Request</td>
<td>Employees submit purchase request.</td>
</tr>
<tr>
<td>$t_2$</td>
<td>Supervisor Approval</td>
<td>The supervisor of task initiator approves the request.</td>
</tr>
<tr>
<td>$t_3$</td>
<td>Fund Checking</td>
<td>When request amount is greater than 1000, a checking is conducted to make</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sure there is sufficient fund.</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Mgmt. Approval</td>
<td>When fund is sufficient, higher level manager needs to approve.</td>
</tr>
<tr>
<td>$t_5$</td>
<td>Purchasing</td>
<td>Purchase Request is reviewed and sent to inventory. PO is requested to be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>printed.</td>
</tr>
<tr>
<td>$t_6$</td>
<td>Print PO</td>
<td>PO is printed.</td>
</tr>
<tr>
<td>$t_7$</td>
<td>Inventory Checking</td>
<td>Inventory checking. If in stock, start shipping; otherwise back order.</td>
</tr>
<tr>
<td>$t_8$</td>
<td>Back Order</td>
<td>The requested item is back ordered.</td>
</tr>
<tr>
<td>$t_9$</td>
<td>Timeout</td>
<td>Timeout is trigger when back order cannot complete after a certain period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of time. A timeout notification is also sent to the purchase requestor.</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td>Shipping</td>
<td>Item is shipped.</td>
</tr>
<tr>
<td>$t_{11}$</td>
<td>Billing</td>
<td>Proper department is billed for the items ordered.</td>
</tr>
<tr>
<td>$t_{12}$</td>
<td>Archiving</td>
<td>The purchase transaction information is summarized and archived.</td>
</tr>
<tr>
<td>$t_{13}$</td>
<td>Disapproval Notification</td>
<td>The requestor is notified that his/her purchase request is declined via email.</td>
</tr>
</tbody>
</table>

Table 1. 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](image)

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.
$p_{13}: t_6, t_9, t_{13}$ 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 policies 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 (WiMC, 1999) as shown in Figure 3.

![Building Blocks for Workflow Modeling](image)

**Figure 3. Building Blocks for Workflow Modeling**

We use $T = \{ t_i | i \in [1,...,n] \}$ 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.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{next}(t_i,t_j)$</td>
<td>task $t_i$ directly leads to task $t_j$</td>
</tr>
<tr>
<td>$\text{fork}(t_i,[t_{i_1}, t_{i_2},...,t_{i_n}])$</td>
<td>task $t_i$ leads to a list of parallel tasks $[t_{i_1}, t_{i_2},...,t_{i_n}]$</td>
</tr>
<tr>
<td>$\text{join}(t_i,[t_{i_1}, t_{i_2},...,t_{i_n}],t_j)$</td>
<td>a list of parallel tasks $[t_{i_1}, t_{i_2},...,t_{i_n}]$ converges to task $t_j$</td>
</tr>
<tr>
<td>$\text{branch}(t_i,[t_{i_1}, t_{i_2},...,t_{i_n}],t_j)$</td>
<td>after task $t_i$ exactly one of the tasks in $[t_{i_1}, t_{i_2},...,t_{i_n}]$ is to be executed</td>
</tr>
<tr>
<td>$\text{merge}(t_i,t_{i_1},...,t_{i_n},t_j)$</td>
<td>any task in $[t_{i_1},t_{i_2},...,t_{i_n}]$ can trigger task $t_j$</td>
</tr>
</tbody>
</table>

**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.

**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. $\text{member}(t_i,[t_{i_1},t_{i_2},...,t_{i_n}])$ means that $t_i$ is a member of the list $[t_{i_1},t_{i_2},...,t_{i_n}]$.

**Definition 1: Task Reachability**

---

Proceedings of the Eleventh Americas Conference on Information Systems, Omaha, NE, USA August 11th-14th 2005 2985
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, \( reach(t_i, t_j) \) 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 \( R = \{ r_i | i \in [1,..,m] \} \) and \( U = \{ u_i | i \in [1,..,ma] \} \) 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.

### Table 3. Organizational Rules

<table>
<thead>
<tr>
<th>Organization Rules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manage((r_i, r_j))</td>
<td>( r_i ) manages ( r_j )</td>
</tr>
<tr>
<td>manage((u_i, u_j))</td>
<td>For all ( u_i \in r_i, u_j \in r_j ) if ( r_i ) manages ( r_j ) then ( u_i ) manages ( u_j )</td>
</tr>
<tr>
<td>role((u_i, r_j))</td>
<td>User ( u_i ) plays role ( r_j )</td>
</tr>
<tr>
<td>manage((r_i, r_k), manage(r_k, r_j))</td>
<td>role privilege inheritance rule:</td>
</tr>
<tr>
<td>manage((u_i, u_k), manage(u_k, u_j))</td>
<td>user privilege inheritance rule:</td>
</tr>
</tbody>
</table>

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 \( u_{ma} \).

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

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.

### Table 4. Task Assignment Rules

<table>
<thead>
<tr>
<th>Task Assignment Rules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r _execute(r_i, t_j) )</td>
<td>Role ( r_i ) is assigned to execute task ( t_j )</td>
</tr>
<tr>
<td>( cannot _r _execute(r_i, t_j) )</td>
<td>Role ( r_i ) is not allowed to execute task ( t_j )</td>
</tr>
<tr>
<td>( u _execute(u_i, t_j) )</td>
<td>User ( u_i ) is assigned to execute task ( t_j )</td>
</tr>
</tbody>
</table>

Figure 5. Definition of Task Reachability

\[
\begin{align*}
&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_j, \{ 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_j, \{ 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)
\end{align*}
\]
must \_u\_execute(u_i,t_j) \quad User \ u_i \ must \ execute \ task \ t_j

cannot\_u\_execute(u_i,t_j) \quad 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) \quad 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) \quad 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) \quad 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:

\begin{align*}
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_2) \leftarrow manage(r_i,r_k),r\_execute(r_k,t_2) \\
p_5 & : \ r\_execute(PC,t_3) \\
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_9),u\_execute(u_{ma},t_9),u\_execute(u_{ma},t_{13})
\end{align*}

Figure 7. Logical Representation of the Workflow Policies

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

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

Table 5. Auxiliary Rules

Some useful queries are explained as follows:

\begin{align*}
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_i,t_j),S_i) & : \ this \ returns \ all \ the \ tasks \ in \ the \ workflow. \ For \ our \ example, \ S_i = \{t_i \mid i \in [1,11]\} \\
manage(r_i,r_j),\ 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} \backslash S_r.
\end{align*}

The union of process rules \ R_p \ 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

\[ \text{findall}(\text{reach}(t_i,t_j),S_t), \text{findall}(\text{role}(u_i,r_j),S_{r+u}), \text{manage}(r_i,r_j), \text{findall}(\text{manage}(r_i,r_j),S_r), S_u = S_{r+u} \setminus S_r \]

A WS is consistent if the following conditions hold:

1. \( \forall t_i \in S_t, \exists u_j \in S_u \text{ such that } \text{count}(u_{\_\text{execute}}(u_j,t_i),n), n \geq 1 \)
2. \( \not\exists u_i \in S_u, \text{ such that } u_{\_\text{execute}}(u_i,t_j) \in WS \text{ and } \not\text{cannot } u_{\_\text{execute}}(u_i,t_j) \in WS \)
3. \( \not\exists r_i \in S_r, \text{ such that } r_{\_\text{execute}}(r_i,t_j) \in WS \text{ and } \not\text{cannot } r_{\_\text{execute}}(r_i,t_j) \in WS \)

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, \( u_{\_\text{execute}}(u_i,t_1) \) and \( \not\text{cannot } u_{\_\text{execute}}(u_i,t_1) \) 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 \( R \) 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

\[ \text{findall}(\text{reach}(t_i,t_j),S_t), \text{findall}(\text{role}(u_i,r_j),S_{r+u}), \text{manage}(r_i,r_j), \text{findall}(\text{manage}(r_i,r_j),S_r), S_u = S_{r+u} \setminus S_r \]

A WS is consistent if the following conditions hold:

1. \( \forall t_i \in S_t, \exists u_j \in S_u \text{ such that } \text{count}(u_{\_\text{execute}}(u_j,t_i),n), n \geq 1 \)
2. \( \not\exists u_i \in S_u, \text{ such that } u_{\_\text{execute}}(u_i,t_j) \in WS \text{ and } \not\text{cannot } u_{\_\text{execute}}(u_i,t_j) \in WS \)
3. \( \not\exists r_i \in S_r, \text{ such that } r_{\_\text{execute}}(r_i,t_j) \in WS \text{ and } \not\text{cannot } r_{\_\text{execute}}(r_i,t_j) \in WS \)
4. \( S = S_t \cup S_u \cup S_r \)
   \[ \text{findall}(r_{\_\text{execute}}(r_i,t_j),S_r) \text{ such that } S_1 \subseteq S \]
   \[ \text{findall}(\not\text{cannot } r_{\_\text{execute}}(r_i,t_j),S_r) \text{ such that } S_2 \subseteq S \]
   \[ \text{findall}(u_{\_\text{execute}}(u_i,t_j),S_u) \text{ such that } S_3 \subseteq S \]
   \[ \text{findall}(\not\text{cannot } u_{\_\text{execute}}(u_i,t_j),S_u) \text{ such that } S_4 \subseteq S \]
   \[ \text{findall}(\text{must } u_{\_\text{execute}}(u_i,t_j),S_u) \text{ such that } S_5 \subseteq S \]

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

\[ \text{findall}(\text{reach}(t_i,t_j), S_i) \]
\[ \text{findall}(\text{role}(u_i,t_j), S_{r+u}) \]

for each logic statement $st_i$ in $R_p$:
execute: \[ \text{findall}(r\_execute(r_i,t_j), S_i), \text{findall}(\text{cannot}\_r\_execute(r_i,t_j), S_2), \text{findall}(u\_execute(u_i,t_j), S_3) \]
\[ \text{compute} \quad S = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 \]

\[ \text{if} \quad S \subseteq S_i \cup S_{r+u} \]
\[ \text{break} \]
\[ \text{elseif} \quad S \cap (S_i \cup S_{r+u}) = \emptyset \]
\[ \text{return REDUNDANT and } st_i \]
\[ \text{exit} \]
\[ \text{elseif} \]
\[ \text{return INVALIDATED, } st_i \text{ and } S \setminus (S_i \cup S_{r+u}) \]
\[ \text{exit} \]
\[ \text{endif} \]
\[ \text{endif} \]
\[ \text{endfor} \]
execute \[ \text{findone} \{r\_execute(r_i,t_j), \text{cannot}\_r\_execute(r_i,t_j)\}, S_6 \]
\[ \text{if} \quad S_6 \neq \emptyset \]
\[ \text{return CONTRADICT and } S_6 \]
\[ \text{exit} \]
\[ \text{endif} \]
execute \[ \text{findone} \{u\_execute(u_i,t_j), \text{cannot}\_u\_execute(u_i,t_j)\}, S_7 \]
\[ \text{if} \quad S_7 \neq \emptyset \]
\[ \text{return CONTRADICT and } S_7 \]
\[ \text{exit} \]
\[ \text{endif} \]
for each $t_i \in S_i$
execute \[ \text{count}(u\_execute(u_i,t_j), n) \]
\[ \text{if} \quad n = 0 \]
\[ \text{return MISSING and } t_i \]
\[ \text{exit} \]
\[ \text{endif} \]
\[ \text{endfor} \]
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 policies, further study of the policies is needed to solve the contradiction, which is out of the scope of this paper. For missing policies, 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.

<table>
<thead>
<tr>
<th>Algorithm 2. Workflow Specification (WS) Consistency Enforcement Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INPUT:</strong> 1) Workflow Specification checked using Algorithm 1.</td>
</tr>
<tr>
<td><strong>OUTPUT:</strong> 1) CHECK and Updated Policy Rules $R_p$ if the algorithm completes successfully, otherwise</td>
</tr>
<tr>
<td>2) FAIL and the task $t_i$ where at least one policy cannot be enforced.</td>
</tr>
</tbody>
</table>

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 $u\_execute(u_j,t_i)$ to $R_p$
    - run Algorithm 1
    - if Algorithm 1 returns SUCCESS
      - return CHECK and exit
    - else
      - remove $u\_execute(u_j,t_i)$ 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.
The new process rules ($R_j$) are shown in Figure 13.

$$next(t_j, t_i), next(t_i, t_2), branch(t_2, [t_3, t_5, t_{13}]), branch(t_3, [t_4, t_{13}]), branch(t_4, [t_5, t_{13}]), branch(t_5, [t_6, t_8]),$$

$$branch(t_8, [t_5, t_9]), next(t_6, t_{10}), merge([t_7, t_{10}], t_{14}), merge([t_{13}, t_{14}], t)$$

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.

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

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.

<table>
<thead>
<tr>
<th>#</th>
<th>Output</th>
<th>Description</th>
<th>Action(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>REDUNDANT and</td>
<td>$p_6$ is redundant because of the deletion of IC and $t_2$</td>
<td>Delete $p_6$</td>
</tr>
<tr>
<td></td>
<td>$p_6 : r _ execute(IC, t_2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>INVALIDATED and</td>
<td>$p_8$ is invalidated because of the deletion of $t_{11}$</td>
<td>Delete $p_8$</td>
</tr>
<tr>
<td></td>
<td>$p_8 : r _ execute(AD, t_{11})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>INVALIDATED and</td>
<td>$p_9$ is invalidated because of the deletion of $t_{12}$</td>
<td>Delete $p_9$</td>
</tr>
<tr>
<td></td>
<td>$p_9 : r _ execute(PC, t_{12})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INVALIDATED and</td>
<td>$p_{11}$ is invalidated because of the deletion of $t_{12}$</td>
<td>Delete $p_{11}$</td>
</tr>
<tr>
<td></td>
<td>$p_{11} : r _ execute(PC, t_{12})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MISSING and $t_{14}$</td>
<td>$t_{14}$ is new and no user is assigned to it.</td>
<td>Assign a machine agent $u_{ma}$ to $t_{14}$,</td>
</tr>
<tr>
<td></td>
<td>$t_{14} : r _ execute(u_{ma}, t_{14})$</td>
<td></td>
<td>Add $p_{13} : u _ execute(u_{ma}, t_{14})$ to $R_p$</td>
</tr>
<tr>
<td>6</td>
<td>SUCCESS</td>
<td>WS is consistent</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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/).

![Figure 15. System Architecture](image)

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