Managing Workflow Processes through Access Control Policies

RASOOL ESMAEILY FARD[1]
ALAN KLEIN[2]
REZA KARIMI NEZHAD[2]

[1]Shiraz University
resmaeily@gmail.com
[2]School of IT, University of Sydney
klein@it.usyd.edu.au, reza.kariminezhad@gmail.com

Abstract: Workflow systems enable organizations to model and execute business processes, but the majority of contemporary workflow management systems are not designed and suited for supporting dynamic business processes. One of the deficiencies is the inability to model realistically the organization of an enterprise to manage the dynamic human-centric business processes. An access control architecture for managing workflow processes is described in the paper. It includes an organizational model and an authorization model for supporting dynamic business processes. More specifically, authorization policies are expressed in an SQL-like language which can be easily rewritten into query sentences for execution. In addition, the architecture supports dynamic integration and execution of multiple access control policies from disparate enterprise resources. Finally, a prototype implementation of the dynamic business process management architecture is described.

Keywords: Workflow process management, Dynamic access control integration, Authorization policy

(Received November 06, 2009 / Accepted May 07, 2010)

1. Introduction

As workflow has been applied to an increasing number of areas, many designs and implementation technologies exist [1]. But, researchers and vendors have been focused mainly on the process logic and IT infrastructure dimensions of workflow and often neglected the organization dimension which consider linkage between the organizational elements and process activities. The complete relationship among the dimensions of workflow and especially the critical role played by the organization dimension are not well studied [2]. However, workflow should support human-centric business processes and therefore must include the modeling of dynamic business roles and human activities. The importance of human involvement in workflow applications has recently been pointed out by [3], who has identified the excessive activity automation and poor design of work assignment strategies as critical issues in workflow projects.

The enforcement of task assignment relies on an authorization model, which is expressed in terms of roles rather than in terms of specific individuals in order to reduce the number of authorizations necessary in the system and to simplify their maintenance [4]. However, this role-based model alone is inadequate to meet all the requirements of processes within an organization. Such requirements may include: (1) role delegation [5], (2) binding of roles [5], and (3) separation of duties [6].

On the other hand, the dynamic business process brings additional challenges to the authorization strategy. For example, as most business processes involve team work, authorization strategy should not only be role based but also be team based [7]. Furthermore, each organization in an enterprise usually enforces its specific management policies; authorization strategies from different management policies should be coordinated.

In this paper an access control model for managing workflow processes is proposed. An access control is specified in a Task Authorization Policy Language (TAPL), which can be easily translated into SQL query sentences so that the access control can be directly executed by a database management system. Based on the TAPL, a policy modeling and enforcement architecture to support dynamic business processes is proposed.

The remainder of the paper is organized as follows. Section 2 gives a brief review of workflow management and introduces a workflow process management architecture. Section 3 presents an organizational model for dynamic business processes. Section 4 defines the syntax of TAPL and discusses the access control modeling and management problem in an organization. Section 5 introduces architecture together with some key techniques to support policy enforcement and access control within a workflow management system. Section 6 describes briefly the implementation of a demonstration system. Section 7 discusses related work. Finally, Section 8 provides some concluding remarks.

2. A workflow process management architecture

There are many process model representations for workflow management implemented by different vendors and proposed by researchers. To facilitate discussion, a brief introduction to a generic process model for workflow management is first given.

A process consists of a set of activities and the dependencies among the activities. The dependencies prescribe the ordering relationships between activities within a process. According to the workflow management coalition (WfMC) [8], six ordering structures may appear in a business process [9]: (1) SEQUENCE—an activity has a single subsequent activity; (2) AND-SPLIT—an activity leads to multiple parallel activities that will all be executed, (3) XOR-SPLIT—an activity leads to multiple but mutually exclusive alternative activities and only one of which will be executed; (4) AND-JOIN—multiple parallel executing activities join into a single activity; (5) XOR-JOIN—multiple but mutually exclusive alternative activities join into a single activity; and (6) LOOP—one or more activities are repeatedly executed until the exit condition is satisfied.

A process can be graphically depicted as a directed graph in which each node represents an activity and each directed edge the dependency [10]. For example, process models are shown in Figure 1, where Figure 1(a) depicts a serial workflow process for software system development and Figure 1(b) is an iterative workflow process model for software component development.

Many approaches have been proposed to improve the adaptability of workflow process to accommodate changes. However, most approaches focus on how to adapt changes for a single workflow, which is inadequate for modeling a dynamic business process that may include tens or hundreds of activities [11].

Researchers have proposed process reuse and activity decomposition as effective ways to support the dynamic business process [12][13]. Figure 2 shows a workflow process management architecture that employs process reuse and activity decomposition. In this architecture, a workflow library stores a set of process models that are designed to meet the business requirements. First, a workflow model is selected to model the entire business process and the high-level ordering constraints of the business process.

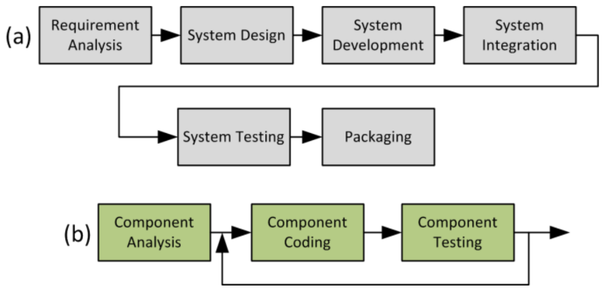


Figure 1. Some examples of process model of workflow. (a) A process model for software system project and (b) a process model for component development.

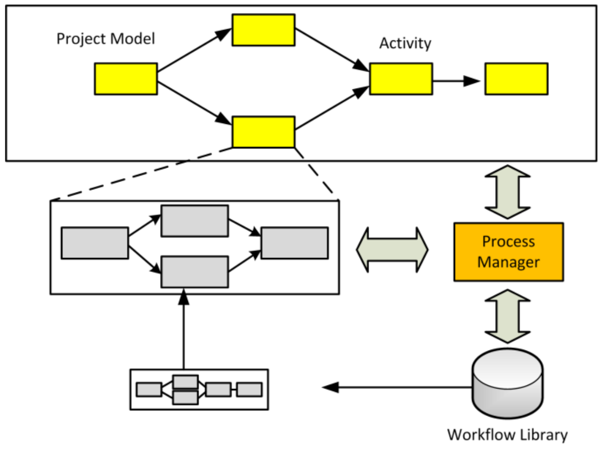


Figure 2. A workflow process management architecture.

Each activity in the project model can be a sub-process, which can be instantiated from another workflow model stored in the workflow library or defined as a set of activities, which can in turn be decomposed further as sub-processes.

For this dynamic process management architecture, an activity may have several child processes. The parent of activity x is obtained by function sup(x). A project p can also be viewed as an abstract activity, in this case, sup(p)=Φ.

In order for this architecture to work, process decomposition and model reuse must be supported. Process decomposition and model reuse rely heavily on the experiences of the project managers and engineers and a knowledge support system is needed to enable decision-making process. A mechanism is also needed to integrate the data objects of a single sub-process instance into the whole business process.

Most importantly, an authorization model needs to be developed to support the dynamic business processes. Next, an organization model for dynamic business processes is described and the development of an authorization model will then be discussed in detail.

3. An organization model for dynamic business processes

The organization model for dynamic business processes is shown in Figure 3. In Figure 3, a project model consists of one or more process models. A process model in turn is composed of a set of activities.

An activity can be complex or atomic. A complex activity includes a set of activities as its children. An atomic activity has no child activities, i.e., for an atomic activity a, ¬x sup(x)=a. When a workflow model is instantiated as a process, an atomic activity should be assigned and it is also called a task. Each activity consists of a set of basic attributes denoted as a={aid, name, type, …}, in which aid, name and type are the unique id, name and type of the activity a. The function type of (a) returns the type of activity a. Inheritance relationships can be defined among activity types. An activity type can be referenced to some process models in the workflow library. For example, an activity type "component development" in a software development project may have reference to two process models. One is "internal development" as depicted in Figure 1(b) and the other could be "outsourced development".

Role is an important concept in the organization model. A role defines a set of capabilities or authorities required to execute certain types of activities. A role can inherit another role. If r_1 inherits r_2, denoted as $r_1 \subset r_2$, role r_1 will subsume all the capabilities or authorities of r_2 a. Role r_2 is called the superior of r_1. This relationship is transitive. A role that has sub-classes is called a virtual role while the role that has no sub-classes is called a concrete role. Figure 4 shows two examples of inheritance relationships between different roles. Role developer can be classified into database developer and user interface developer and similarly, role tester can be classified into database tester and user interface tester.

A staff in an organization is defined with a set of basic attributes, denoted as m={id, $attr_1$, $attr_2$, …}. Each staff can be assigned several concrete roles reflecting the abilities of the staff person. The relationship play (m, r) represents that the staff m can play the role r. A staff belongs to a department, which includes a role set as its structure definition.

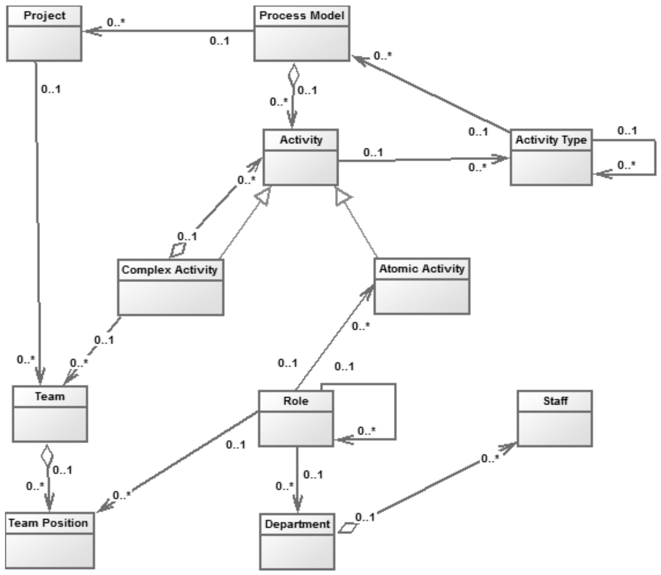


Figure 3. An organization model for dynamic business process.

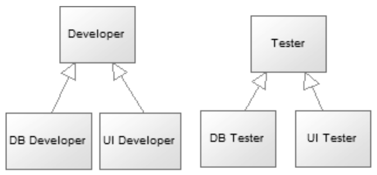


Figure 4. Role hierarchy.

For a given project, teams are put together to deal with various business tasks. The members of a team may be assembled from several departments temporarily for specific tasks. A team consists of several team positions, which define the role requirements for the team. Staffs can be selected and assigned to hold these positions. It is assumed that the staff members assigned are capable to play the roles required by these positions. A team can thus be formally represented by T= ⟨PS, RS, MS, RM⟩, where PS represents the position set, RS is a role set, MS represents the members in this team, and RM=PS×RS×MS corresponds to the assignments and positions of the members. For example, a design team can be represented as t= ⟨{leader, member}, {manager, UI developer, UI tester}, {Esmaeily, Mehdipour, Hassani, Ahmadi}, {⟨leader, manager, Hassani⟩, ⟨member, UI developer, Esmaeily⟩, ⟨member, UI developer, Mehdipour⟩, ⟨member, UI tester, Ahmadi⟩}⟩. Member can play one or more roles in a team. Finally, the functions team_member (t), team_role (t) and enabled_role(t, s) are designed to retrieve the staff members of the team, the role set of the team and the role set that the staff s plays in the team t.

4. An authorization model for dynamic business processes

There are different task authorization strategies that can be deployed for task assignment in a business process. The four most basic task authorization types are:
(1) Staff-authorization: to assign a staff for a task.
(2) Role-authorization: to assign a specific role for a task.
(3) Team-authorization: to form a team and assign the task to the team. A team can further be divided into sub-teams. If a team t_1 is a sub-team of team t, then $(\text{team\_member}(t_1) \subseteq \text{team\_member}(t)) \wedge (\text{team\_role}(t_1) \subseteq \text{team\_role}(t)) \wedge (\forall s \in \text{team\_member}(t_1), \text{enabled\_role}(t_1, s) \subseteq \text{enabled\_role}(t, s))$ must hold. Each team is responsible for a complex activity and the project team deals with the whole project. If there is no team assigned to an activity directly, then the function sup() defined in Section 2 can be issued several times till an activity that has a team assigned to is found so that each activity in the process corresponds to a team and the team of an activity a is obtained by function teamof (a).
(4) Department-authorization: in some cases, an activity is assigned to a specific department or division to maintain the autonomy of an organization. The department has the responsibility to determine its task assignments. This activity can be considered as a sub-project, in which task assignment strategy is handled autonomously within the department.

A project may employ all four type authorization strategies. A Task Assignment Policy Language (TAPL) is developed to describe the task authorization strategies and to represent complex requirements in task allocation.

4.1. TAPL—Task Assignment Policy Language

TAPL is a policy language with simple syntax but is adequate to express complex constraints in the task authorization for a business process.

The syntax of TAPL is shown in Table 1. In TAPL, a "*when*" clause includes a group of pre-defined functions such as *IsFull(), IsAssigned()* and *Play().* They can be used to retrieve the history of task allocations and evaluate current situations. Furthermore, "*" represents "all" and "#" represents "*exist*". The "*where*" clause is applied to refine the roles defined in the policy. It includes a set of ranges and functions. The function used in "*where*" clause of current version TAPL is *HasSkill(),* which represents that a role having some specific skills is needed. The "*with*" clause includes a set of ranges to further specify the activity type that a policy states. Figure 5 shows an example of task authorization policies related to a process instance "AM Component Development".

Table 1. The syntax of TAPL

statement	::= ⟨ require ⟩ | ⟨ substitute ⟩ | ⟨ reject ⟩
require	::= require ⟨ resource ⟩ ⟨ where ⟩ ⟨ when ⟩ ⟨ for ⟩ ⟨ with ⟩
substitute	::= substitute ⟨ resource ⟩ ⟨ where ⟩ ⟨ when ⟩ by ⟨ resource ⟩ ⟨ where ⟩ ⟨ for ⟩ ⟨ with ⟩
reject	::= reject ⟨ resource ⟩ ⟨ where ⟩ ⟨ when ⟩ ⟨ for ⟩ ⟨ with ⟩
for	::= for ⟨ activity ⟩ | ⟨ activity_type ⟩
activity	::= activity ⟨ activity_id ⟩ ⟨ activity_type ⟩ ::= activity_type ⟨ activity_type_id ⟩
resource	::= * |#| ⟨ person ⟩| ⟨ role ⟩
person	::= person ⟨ person_id ⟩
role	::= role ⟨ role_id ⟩
when	::= ⟨ empty ⟩ | when ⟨ functions ⟩
where	::= ⟨ empty ⟩ where ⟨ ranges ⟩|where ⟨ functions ⟩|where ⟨ functions ⟩ AND ⟨ ranges ⟩
with	::= ⟨ empty ⟩ | with ⟨ ranges ⟩
ranges	::= ⟨ range ⟩|⟨ range ⟩ AND ⟨ ranges ⟩
range	::= ⟨ attribute ⟩ ⟨ op ⟩ ⟨ value ⟩
op	::= > | < | = | >= | <=
functions	::= ⟨ function ⟩ and ⟨ functions ⟩

In TAPL, a policy is divided into three types, namely the requirement policy, the scenario policy and the substitution policy.

4.1.1. Requirement policy

A requirement policy defines the required roles for a task. The name of a specific activity or an activity type is defined by a "*for*" clause which can be further specified by a "*with*" clause. Constraint conditions for a role can be specified in a "*where*" clause. If several requirement policies are specified for an activity and the role defined in these policies are the same, then the selected staffs should meet all the constraints defined in the "*where*" clauses of these policies. For the example shown in Figure 5, on the activity "AM analysis", policies 1–3 refer to the same role, i.e., "analyst".

4.1.2. Scenario policy

A scenario policy can be specified to restrict certain people to be assigned to a specific activity or an activity type. If there are more than one scenario policies defined for one activity, then the policies should be considered altogether. For the example shown in Figure 5, policy 9 illustrates a situation that no further tasks should be assigned to a staff when the working load of the staff is full.

4.1.3. Substitution policy

This policy states that if roles (staffs) defined by requirement policies cannot be found, the roles can be substituted by other roles. If there are more than one policy defined for the same activity, then the policies represent different ways to find substitution roles. For example, policy 8 states that the staff who is qualified for playing the role "UI Developer" and has skill "database programming" can also play the role of "DB Developer" if necessary.

1. **require** role "Analyst" **for** activity "AM Analysis"
2. **require** role "Analyst" **where** experience>=3 **for** activity "AM Analysis" **with** DifficultDegree>4
3. **require** role "Analyst" **where** HasSkill("Rational Rose") **for** activity "AM Analysis"
4. **require** role "Developer" **where** experience>=5 **for** activity "AM Analysis" **with** DifficultDegree<6
5. **require** role "DB Developer" **for** activity "AM Coding" **with** MainTechnology = "Database"
6. **require** role "DB Tester" **for** activity "AM Testing" **with** MainTechnology="Database"
7. **reject** role "Tester" **when** AssignedTo(activity "AM Coding") **for** activity "AM Testing" **with** NumberOfLines>500
8. **substitute** role "DB Developer" **by** role "UI Developer" **where** HasSkill("Database Programming")
9. **reject** * **when** IsFull('*') **for** *

Figure 5. An example of task authorization policy.

4.2. Modeling and management of task authorization policy

The task authorization policies are closely related to the management strategies of an organization. Policies can be specified in different management levels and for different scopes. Since an organization may have many authorization policies defined, a modeling and management architecture is needed to systematically specify task authorization policies.

Task assignment policies can be stored in a library. A task authorization policy can be represented as PL = ⟨pid, Con, St, Sc⟩, where pid, Con, St, Sc are its identification, the content defined in TAPL, its status and the scope, respectively. Assignment policies can be categorized into four types according to the scope as shown in Figure 6:

(1) *Department policy*: Each department can have its own task authorization policy. After accepting an activity, this department can define its own sub-process and allocate tasks.

(2) *Process policy*: Process policy is attached to each process model in the workflow library. A process policy can be an activity policy or coordination policy. An activity policy is defined for each activity or activity type specifically. A coordination policy defines task assignment relationships among the activities.

(3) *Project policy*: Each project can have its own policies, which applies to all the tasks in the project. For example, a project policy:

reject * when IsFull("*") **for** *

denotes that it is not permitted to assign extra tasks to any person whose workload is full.

(4) *Team policy*: Project or team managers can specify their specific task assignment policies for the whole team. For example, for a team assigned to the activity "system development" for the software project, the team may be imposed a substitution policy:

substitute role "DB Developer" **by** role "UI Developer" **where** HasSkill("Database Programming")

which indicates a "DB Developer" can be substituted by a "UI Developer" if appropriate skill is met. The policy will also apply to all the tasks on the two sub-processes, i.e., UI component development and AM component development.

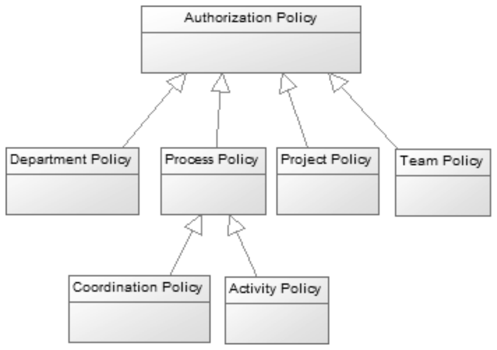


Figure 6. Different authorization policies in an organization.

5. Policy enforcement for task assignment

5.1. A task assignment architecture

Figure 7 depicts a task assignment architecture for dynamic business processes. Before a project begins, a project manager can form a project team according to the knowledge about the project. The project manager can also add certain task assignment policies to the project and to the project team. As activities are decomposed into sub-processes, sub-teams and their team policies can be established. A sub-team manager can further append policies to the activities or tasks. For example, the sub-team manager may add a policy to the activity "system development" as follows:

require role "Analyst" **where** HasSkill("Rational Rose") **for** activity_type "Component Analysis"

According to the organizational requirements, activities can be directly assigned to specific departments.

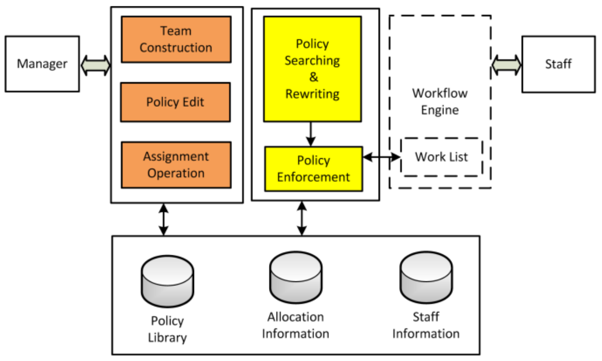


Figure 7. A task authorization architecture for dynamic business processes.

Task execution is handled by a workflow engine. The workflow engine sends a request to the policy search and rewriting module, which retrieves all related authorization policies for a task and rewrite these policies into executable clauses. The policy enforcement module then executes these clauses to find a set of qualified staffs for the task and output the results as a work list managed by the workflow engine. All candidates will then receive a task notification from the workflow engine. The task will be assigned to the candidate who accepts the task and no other candidate will further be assigned unless a manager intervenes in the process by assigning another candidate to the task directly.

5.2. Authorization policy search

With the existence of different authorization policy sources, a mechanism is needed to search all the relevant policies in order to fulfill a task assignment correctly. Figure 8 shows the flow chart of an authorization policy search algorithm for a given task. Because each task is instantiated from a workflow process model, the first two steps involve collecting all the related activity policies and coordination policies from the workflow process model. Other policies are then added according to the team structure and project process structure. If an activity has a parent and a team is directly assigned to the parent, then all team policies should be selected for the task. The process policies defined in its parent activity will also be collected for this task. This process continues until the current pre-defined project model is reached. Project policies and team policies to the task are added. During the search process, if an activity is allocated to a department explicitly, then department policies should be considered and supercede the project policies.

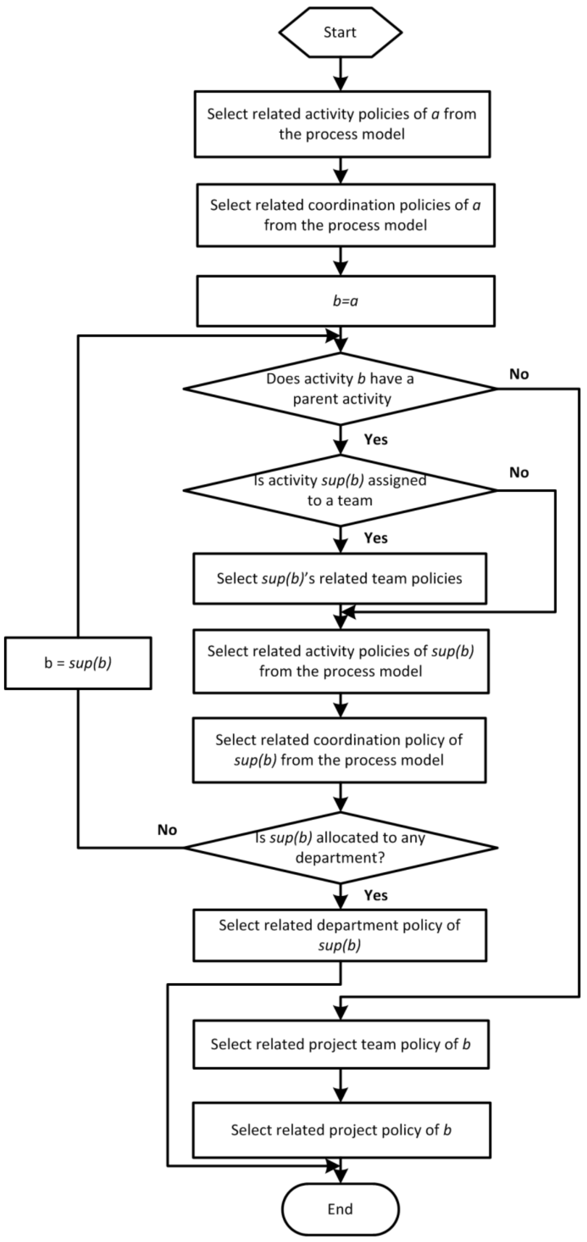

Start
Select related activity policies of a from the process model
Select related coordination policies of a from the process model
b=a
Does activity b have a parent activity
No
Yes
Is activity sup(b) assigned to a team
No
Yes
Select sup(b)'s related team policies
b = sup(b)
Select related activity policies of sup(b) from the process model
Select related coordination policy of sup(b) from the process model
No
Is sup(b) allocated to any department?
Yes
Select related department policy of sup(b)
Select related project team policy of b
Select related project policy of b
End

Figure 8. An algorithm of authorization policy search for a task

An important issue in the search algorithm is to determine if a policy is related to a task. The following rules are employed to determine for the relevancy:
(1) The activity type or the id of the task should be consistent with the content of the "*for*" clause, i.e., the type of the task should be defined by the "*for*" clause or is a sub-type of that defined by the "*for*" clause.
(2) If a "*with*" clause exists in a policy, the properties of the task are checked with the constraints defined in the clause. If the constraints can be satisfied, the policy will be included, otherwise the policy will be ignored.
(3) If a policy is not related to any activity type or a specific activity, then the policy is included only if the

role defined in the policy is equal or superior to a role defined in other selected requirement policies for the task.

Finally, if the content in "*for*" clause is an activity type, then it will be replaced by the id of this task.

For example, if the "DifficultyDegree" of the task "AM Analysis" in a specific process is greater than 4 and less than 6, task policies for the task "AM Analysis" are shown in Figure 9.

5.3. Policy rewriting and enforcement

After the authorization policies for a task a are obtained, these policies are executed to find qualified staffs from *teamof(a).* The requirement and scenario policies are executed first. When no qualified staffs are found based on requirement and scenario policies, substitution policies are then executed. In this work, the policies are translated into SQL query sentences, which can then be executed by a database management system (DBMS) directly.

1. **require** role "Analyst" **for** activity "AM Analysis"
2. **require** role "Analyst" **where** experience>=3 **for** activity "AM Analysis" **with** DifficultDegree>4
3. **require** role "Analyst" **where** HasSkill("Rational Rose") **for** activity "AM Analysis"
4. **require** role "DB Developer" **where** experience>=5 **for** activity "AM Analysis" **with** DifficultDegree<6
5. **reject * when** IsFull("*") for activity "AM Analysis"
6. **substitute** role "DB Developer" **by** role "UI Developer" **where** HasSkill("Database Programming")

Figure 9. Policies for the task "AM Analysis" of a specific process.

In the TAPL, the "*for*" and "*with*" clauses are used for the policy search purpose; there is no need to translate them into the SQL clauses. The "*where*" and "*when*" clauses act as filters and they are mapped into "*select*" sub-clauses conforming to the SQL syntax. The functions applied in the "*where*" and "*when*" clauses can be translated into "*select*" sub-clauses according to the pre-defined templates. For a policy *p* defined for an abstract role *r*, if a role defined in another policy is the sub-role of *r* then *p* should be translated into the policy acting on this sub-role. If there is no any other policy defined to the sub-role of *r*, then *p* should be translated into policies acting on all of its concrete roles. This process continues recursively until all roles defined in the policies are all concrete roles.

The following example illustrates how policies are rewritten into SQL sentences. Suppose the relational tables are defined as follows (with keys underlined):

• resource(resource_id, experience, workingload, …)
• team_member(team_id, resource_id, role_id, …)
• resource_skill(resource_id, skill, …)
• allocated_task(activityid, resource_id, …)
• role (role_id, rolename, …)

Let the team_id of *teamof(a)* as ID. The procedures for rewriting policies into SQL sentences are shown below.

5.3.1. Rewriting requirement policy

The rule for rewriting requirement policies into a SQL query sentences is shown in Figure 10.

As an example, for the activity "AM Analysis" shown in Figure 8, policies 1–3 are all related to the role "Analyst" and policy 4 is related to the role "DB Developer". The requirement policies can be rewritten into:

```
"SELECT a.resource_id FROM resource as a,
team_member as b, resource_skill as c,
allocated_task as d, role as e
WHERE ((e.rolename='Analyst' AND
e.role_id=b.role_id) AND ((a.experience>=3)
AND(c.skill='RationalRose' AND
c.resouce_id=a.resource_id))) OR
((e.role='DBDeveloper' AND
e.role_id=b.role_id) AND (a.experience>=5)
AND (b.team_id=ID AND
b.resource_id=a.resource_id))"
```

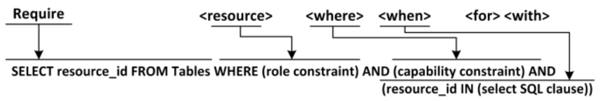

Require
<resource>
<where>
<when>
<for> <with>
SELECT resource_id FROM Tables WHERE (role constraint) AND (capability constraint) AND
(resource_id IN (select SQL clause))

Figure 10. Rewrite a requirement policy to a SELECT sentence in SQL.

5.3.2. Rewriting scenario policy

Figure 11 shows the rewriting rule for the scenario policy, which is used to eliminate the staffs from those selected by requirement policies.

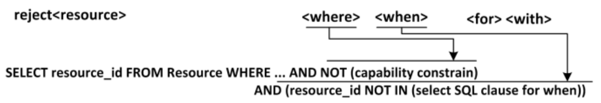

reject<resource>
<where>
<when>
<for> <with>
SELECT resource_id FROM Resource WHERE ... AND NOT (capability constrain)
AND (resource_id NOT IN (select SQL clause for when))

Figure 11. Rewrite a scenario policy to a SELECT sentence in SQL.

As for the activity "AM Analysis" shown in Figure 9, policy 5 states that no task should be allocate a staff whose working load is full. The SELECT sentence can be rewritten to:

```
"SELECT a.resource_id FROM resource as a,
team_member as b, resource_skill as c,
allocated_task as d, resource_role as e WHERE
((e.role='Analyst' AND e.role_id=b.role_id)
AND ((a.experience>=3) AND
(c.skill='RationalRose' AND
c.resouce_id=a.resource_id) )) OR
```

```
((e.role='DBDeveloper' AND
e.role_id=b.role_id) AND (a.experience>=5)
AND (b.team_id=ID AND
b.resource_id=a.resource_id)) AND (
a.resource_id NOT IN(SELECT resource_id FROM
resource WHERE resource.workingload>=8))"
```

In the query sentence, "SELECT resource_id FROM resource WHERE resource.workingload>=8" is generated from the mapping template defined for function IsFull().

6. Implementation

A workflow management system for dynamic business process has been implemented based on the architecture shown in Figure 2. Access control modeling and enforcement modules are two important parts of the whole system.

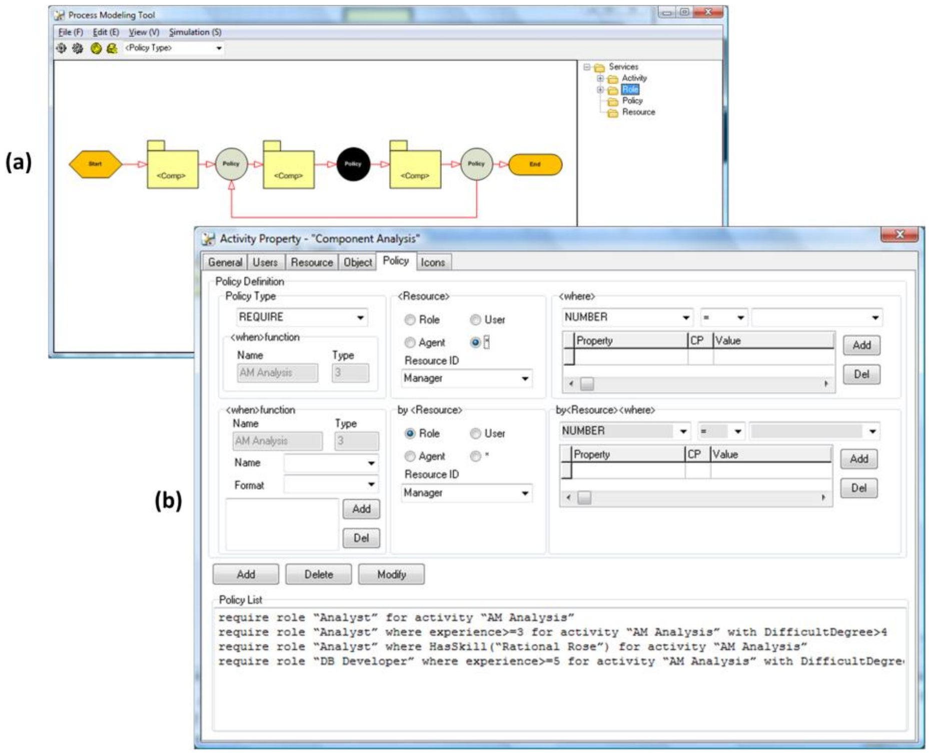
Figure 12(a) is a process-modeling environment through which a workflow process model can be defined and saved into a workflow library. In the environment, each workflow model is represented as a process graph. The model shown in Figure 12(a) is the component development process introduced in the paper. By doubling click the activity node of the graph, the properties such as activity description, resources allocation, input and output objects of the selected activity can be edited. Specifically, the environment provides an interface for defining task assignment policies as shown in Figure 12(b). This interface is a policy editor through which a modeler can add policies piece by piece. Figure 12(b) shows the example of policy definition for the activity "component analysis".

A workflow engine and a task assignment enforcement module have been developed using the mechanism shown in Figure 7. User interfaces, which are developed based on the outlook web access of Microsoft Exchange server 2000, are provided to staffs and managers. A staff can access his work list to receive a task request or submit a task through the interface. Managers can monitor the process, allocate staffs to specific tasks and create sub-processes using the interface.

7. Related work

It is widely accepted processes are the core of organizations [14], organizations also have important impact on process. Organizational models for workflow management have been proposed by [2][15][16]. But they only defined some Meta models of the organization structure for workflow management, which can only serve as a basis for task assignment research.

Most of the researches in recent years regard role-based model as a set of constraints[17]. Constraints can be static or dynamic and they can be applied to a whole class of processes, or to specific instances. Mechanisms for constraint specification range from logic languages [17] to Petri Nets [18]. Constraint-based



Policy Type: REQUIRE
<when>function Name: AM Analysis Type: 3
<Resource> Resource ID: Manager
<where>: NUMBER =
<when>function Name: AM Analysis Type: 3
by <Resource>: Role; Resource ID: Manager
by<Resource><where>: NUMBER =

```
require role "Analyst" for activity "AM Analysis"
require role "Analyst" where experience>=3 for activity "AM Analysis" with DifficultDegree>4
require role "Analyst" where HasSkill("Rational Rose") for activity "AM Analysis"
require role "DB Developer" where experience>=5 for activity "AM Analysis" with DifficultDegre
```

Figure 12. A process modeling environment supporting task authorization policy modeling.

method employs algorithms to check the consistency of constraints and assign users and roles to the tasks that constitute the workflow in such a way that no constraints are violated. Although powerful, constraint-based method suffers the complexities brought by its representation and consistency checking process.

In [4] authorization constraints are expressed as event–condition–Action (ECA) rules. The event part denotes when an authorization may need to be modified. The condition part verifies that the occurred event actually requires modifications of authorizations, and determines the involved agents, roles, tasks and processes. The action part enforces authorizations and prohibitions. This authorization model does not take into consideration the properties of process and staffs on task assignment. EROICA framework [5] extends the syntax of the ECA rules, but it does not provide an organizational rule modeling and enforcement architecture for dynamic business processes.

In order to add a team concept to current workflow management systems, the Object Constraint Language (OCL) to define the relationships among persons is proposed [19]. OCL is part of the UML language to describe the relationship among classes. Since OCL provides a modeling mechanism to the static relationship among classes and objects, it can be used to define the structure of a team. However, OCL is complex and non-descriptive. Processing OCL for task assignment requires a special parser.

Bussler is the first researcher who proposed a policy-based task assignment architecture [20]. It is further be expanded to the resource query language RQL proposed by HP lab [21]. The access control language described in the paper is quite similar to the RQL. RQL is a SQL-like language and is able to specify three types of policies: qualification, requirements and substitution policies. The functions of requirement and substitution policies are similar to the ones described in the paper. A qualification policy is used to state the type of resources, which is qualified to do an activity type. TAPL provides a few new features. First, a "*when*" clause is provided to represent current allocation status and process status. Second, functions are introduced to express complicated resource allocation conditions. Third, a new type policy, i.e., scenario policy is introduced to confine the search scope according to some important criteria, for example, the separation of duty in workflow. In addition to these differences in the policy language, we also present an architecture for organizational access control modeling and enforcement to support dynamic business processes. Furthermore, a policy search algorithm and enforcement methods are developed to adapt to the features of dynamic business processes such as team work and dynamic decomposition of activity.

Recent years, to make extensions for the industry process model languages such as BPEL4WS and Business Process Modeling Notation (BPMN) to express task authorizations becomes a research focus. For example, in [22] formal architecture that integrates RBAC into BPEL and allows expressing authorization constraints using temporal logic is presented. In this architecture model-checking can be applied to verify that a given BPEL process satisfies the security constraints. Although it can make use of available model-checking tools for constraint satisfaction check, the common users can not apply temporal logic to represent related authorization constraints directly. In [23], an extension for the BPMN to express authorizations within the workflow model is proposed. It enables the support of resource allocation pattern, such as separation of duty, role-based allocation, case handling, or history-based allocation in BPMN. Comparing with their work, our architecture supports access control modeling in different scope and the enforcement of policies for dynamic business process are also provided while this topic is not covered in [23].

In [24], multi-criteria assessment model capable of evaluating the suitability of individual workers for a specified task according to their capabilities, social relationships, and existing tasks has been proposed. Candidates are ranked based on their suitability scores to help administrators to select qualified workers to perform the tasks assigned to a given role. The task assignment policy described in this paper focuses on the role-assignment for a task while at the same time defines the specific requirements for a role based on either workers' capabilities or process properties. The result can be the input into a multi-criteria assessment model for selecting qualified staffs.

[25] discuss workflow management and verification and validation issues where authorization control is also an important issue. But the authorization issue is stated simply there without further investigation.

8. Conclusions and future work

There is a need to develop tools and models for supporting dynamic business processes. This paper focuses on providing an effective task assignment strategy for dynamic business processes. An architecture is proposed to support dynamic access control modeling and enforcement in a business process environment where assignment policies come from different sources.

The mechanism for facilitating access control is based on an SQL-like language called TAPL, which can be rewritten into SELECT sentences in SQL. TAPL can describe complex role constraints in a business process. Using standard SQL technology eliminates the need for developing a complex parser and executing components.

In the current version of TAPL, a particular function is interpreted as an SQL sub-clause by a template that is developed as a part of pre-defined TAPL transforming program. A scalable TAPL transforming program architecture that allows functions to be added and templates integrated into the architecture should be investigated in the future.

References

[1] Becker, J., zur Muehlen, M., 2002. "Workflow application architectures: classification and characteristics of workflow-based information systems". *Workflow handbook*, pp. 39–50.

[2] Zur Muehlen, M., 2004. "Organizational management in workflow applications—issues and perspectives", *http://www.workflow-research.de/Publications/PDF/MIZU-ITM (2004).pdf.*

[3] Moore, C., 2002. "Common mistakes in workflow implementations, *Giga Information Group RIB-062002-00118"*, Cambridge, MA.

[4] Casati, F., Castano, S. and Fugini, M.G., 2001. "Managing workflow authorization constraints through active database technology", *Journal of Information Systems Front (Special Issue on Workflow Automation and Business Process Integration),* 3 (3), pp. 319–338.

[5] Akhil, K. and Zhao, L.J, 2002. "EROICA: A rule-based approach to organizational, policy management". *Workflow systems, WAIM 2002*, vol. 2419 pp. 201–212.

[6] Botha, R.A., Eloff, J.H.P., 2001. "Separation of duties for access control enforcement in workflow environments", *IBM System Journal* 40 (3), pp. 666–681.

[7] van der Aalst, W.M.P., 2001. "A reference model for team-enabled workflow management systems", *Data Knowledge Engineering*, 38 (3), pp. 335–363.

[8] Workflow Management Coalition, 2004. *http://www.wfmc. org*

[9] Workflow Management Coalition, 1999. WFMC-TC-1011, *http://www.wfmc.org/standards/docs/TC-1011_term _glossart _v3.pdf,*

[10] Duenren, L. and Minxin, S., "Workflow modeling for virtual processes: an order-preserving process-view approach", *Information Systems*, 28 (6), 2003, pp. 505–532.

[11] van der Aalst, W.M.P. and Jablonski, S., 2000. "Dealing with workflow change: identification of issues and solutions", *Computer Systems Science and Engineering* 15 (5), pp. 267–276.

[12] Chung, P.W.H., Cheunga, L. and Stader, J., 2003. "Knowledge-based process management—an approach to handling adaptive workflow", *Knowledge-Based Systems*, 16 (3), pp. 149–160.

[13] Myungjae, K., Dongsoo, H., Jaeyong, S., 2002. "A framework for dynamic workflow interoperation using multi-subprocess task". *In Proc. of the 12th international workshop on research issues in data engineering: engineering e-commerce/e-business systems (RIDE.02)*, pp. 99–130.

[14] Willaert, P., Van den Bergh, J., Willems, J., Deschoolmeester, D., 2007. "The process-oriented organisation: a holistic view developing a framework for business process orientation maturity", *Business process management, 5th international conference, BPM 2007, Proceedings, lecture notes in computer science, vol. 4714,* Brisbane, Australia, September 24–28, pp. 1–15.

[15] Qiu, J., Ma, C., 2008. "A flexible access control model for workflows". In Proc. *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on 7,* Xian, China, pp. 606-612.

[16] Bussler, C., 1998. "Organisationsverwaltung in workflow- management-systemen", *Deutscher Universit.ts-Verlag,* Wiesbaden, Germany.

[17] Bertino, E., Ferrari, E., 1999. "The specification and enforcement of authorization constraints in workflow management systems", *ACM Transaction on Information System and Security* 2 (1), pp. 65–104.

[18] Atluri, V., Wei Kuang, H., 2000. "A petri net based safety analysis of workflow authorization models", *Journal of Computer Security 8 (2/3),* pp. 209–240.

[19] van der Aalst, W.M.P. and Jablonski, S., 2000. "Dealing with workflow change: identification of issues and solutions", *Computer Systems Science and Engineering* 15 (5), pp. 267–276.

[20] Bussler C, and Jablonski, S., 1995. "Policy resolution for workflow management systems"., *28th Hawaii international conference on system sciences, hicss*, pp. 831–840.

[21] Yan-Nong, H., Ming-Chien, S., 1998. “Policies in a resource manager of workflow systems: modeling, enforcement and management”, HPL-98-156, *http://www.hpl.hp.com/ techreports/98/HPL-98-156.pdf*.

[22] Xiangpeng, Z., Cerone, A., Krishnan, P, 2006. “Verifying BPEL workflows under authorization constraints”. *Business process management, 4th international conference, BPM 2006, , Proceedings, lecture notes in computer science, vol. 4102,* Vienna, Austria, September 5–7, pp. 439–444.

[23] Lu, Y., Zhang, L., Sun, J., 2009. “Task-activity based access control for process collaboration environments”. *Computers in Industry*, Elsevier Science. In Press.

[24] Minxin, S., Gwo-Hshiung, T., and Duen-Ren, L., 2003. “Multi-criteria task assignment in workflow management systems”. *In Proc. of the 36th Hawaii international conference on system sciences (HICSS’03),* p.p. 202–210.

[25] Chen, J., Yang, Y., 2007. “Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in grid workflow systems”. *ACM Transaction on Autonomous Adaptive Systems*, 2(2), Article 6.