RC24523 (W0803-123) March 28, 2008
Computer Science

IBM Research Report

A Framework Based on Role Patterns to Design Secure Business Processes

Akhil Kumar
Smeal College of Business
Penn State University
University Park, PA 16802

Rong Liu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home .

A Framework based on Role Patterns to design Secure Business Processes

Akhil Kumar[a] and Rong Liu [b]

[a] Smeal College of Business, Penn State University, University Park, PA 16802, USA
[b] IBM Research, 19 Skyline Drive, Hawthorne, NY 10532, USA
akhil@psu.edu, rliu@us.ibm.com

Abstract. In view of recent business scandals that prompted the Sarbanes-Oxley legislation, there is a greater need for businesses to develop systematic approaches to designing secure business processes where the security aspects can be integrated into the process tightly. In this paper we propose 10 role patterns, and show how they can be associated with generic task categories and processes in order to meet standard requirements of internal control principles in businesses. We also show how the patterns can be implemented using built-in constraints in a logic based language like Prolog. While the role patterns are general, this approach is flexible and extendible because user-defined constraints can also be asserted in order to introduce additional requirements as dictated by business policy. The paper also discusses control requirements of business processes, explores the interactions between role based access control (RBAC) mechanisms and workflows, and gives an architecture for integrating our framework into an existing workflow system.

1. Introduction

Sarbanes-Oxley legislation in the United States and similar laws in other countries have highlighted the importance of making business processes secure. It has made it mandatory for top officers of organizations to certify that suitable controls are in place to guarantee that processes are secure. Section 302 of Sarbanes Oxley Act [13,14] requires that CEOs and CFOs must personally sign off on their company's financial statements, while Section 404 requires that appropriate processes and control must exist for all aspects of a company's operations that affect financial reports.

In this paper we discuss what it means for business processes to be secure and discuss ways of ensuring such security. The requirements of an accounting application might state that: (1) an invoice must be approved before it is paid; (2) the goods must be received before the invoice is approved for payment; and, (3) the goods must be inspected before the payment is made. Similar needs arise in applications in patient care, immigrant processing, insurance claims, etc.

We first consider ways in which security can be breached, and describe a framework that can be used to prevent such breaches. The framework consists of three parts: *task categories, process patterns, role patterns* and *constraints*.

A process pattern or workflow [1] is a generic pattern that specifies the ordering of tasks and subprocesses required for performing well-known functions in a business. Process patterns are quite general and apply in a variety of domains. Thus, a generic 'Order' process may be applied in various applications such as:

- Place an order for a laptop computer
- Request a service like a flight booking or hotel reservation
- Obtain a new computer account
- Receive permission or authorization for paid leave
- Submit a new loan application

There are certain basic combinations of task categories that can be combined to create an *Order* process. Every order has to be prepared; It has to be approve; it has to be submitted; it has to be received; it has to be paid for; etc.

The second aspect of the framework is *role patterns*. *Roles* are standard designations or titles on the organization chart of any company. Role patterns are a way to restrict the roles that can participate in a process instance both in terms of the sequence in which the role can participate, and the number of times the role can participate. For instance, an employee may be allowed to fill in the travel budget for a business trip, but after it is approved, the employee is not allowed to change any amounts.

The third feature of the framework is constraints, both built-in and user-defined. There are various types of constraints in any business process, such as separation of duties constraints, binding of duties, and other types of constraints. Some standard constraints are captured in the task and rule patterns; however, more specialized and fine-grained constraints can also be added using this mechanism.

The goal of this paper is to show how processes can be made secure by integrating the various elements of our framework and to develop a methodology for designing secure business processes. We present a framework with three dimensions: process patterns, role patterns and constraints. We argue that taken together these dimensions can allow us to create secure processes.

This paper progresses as follows. Section 2 provides basic background on the modeling and design of business processes. Section 3 covers basic principles of internal control in a business. Next, Section 4 discusses role patterns and gives a detailed example to illustrate the implementation of our approach. Then, Section 5 provides a discussion along with related research. Finally, Section 6 concludes the paper.

2. Preliminaries

A business can be viewed as a collection of processes, and the robustness of these processes to a large extent is a crucial determinant of the success of the business. Business processes can be described using some simple constructs, and most workflow application system products provide support for these constructs. Four *basic constructs* that are used in designing processes are *immediate sequence*, *decision structure* and *loop*, as shown in Figure 1.

In general, business processes can be composed by combining these four basic patterns as building blocks. They can be applied to atomic tasks, e.g. Iseq(A,B) to indicate that tasks A and B are combines in parallel, or to subprocesses, e.g. Iseq(SP1,

SP2) to indicate that subprocess SP1 and subprocess SP2 are combined in sequence. In Figure 1(a), we use ISeq to indicate two tasks or subprocesses are in immediate sequence. Parallelism is introduced using AND-Split control nodes and parallel branches can be synchronized by AND-Join nodes at the end as shown in Figure 1(b). A choice structure is shown in Figure 1(c). In this pattern, this first OR construct, called OR-split, represents a choice or a decision point, where there is one incoming branch and it can activate any one of the two outgoing branch. The second OR construct is called an OR-Join because two incoming branches join here and there is one outgoing branch. Finally, it is also possible to describe loops in a process diagram by combining a pair of OR-split and OR-join nodes, such that one outbranch from the OR-Split node connects to an in branch of an OR-join node, as shown in Figure 1(d). In this way, the patterns can be applied recursively to create a complete process.

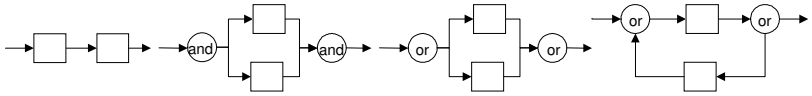


Figure 1: Basic patterns to design processes

As a running example for this paper, Figure 2 shows an example of an "Administer Account Transfer" process. It starts with a **customer representative** receiving an account transfer instruction from a client. A **financial clerk** then checks if the details of this transfer instruction are complete, and, if so, gives an affirmative reply. If the instructions are incomplete, a communication details regarding the invalid payment instruction are extracted. If the payment instruction is accepted, the transaction limit is checked by a **financial accountant** and if it is within the limit, then the funds availability is tested by a **banking specialist**. If the transaction limit is exceeded, a request for authorization is made. If this request is approved by the finance manager, then the transaction proceeds normally, i.e. appropriate accounting entries are created and applied to the required accounts, communication details are extracted from the accounting entries and the customer is notified. Otherwise, the operational risk in the transaction is evaluated by the **risk analyst**. If the risk is small a **risk manager** may approve the transaction so it can proceed. Otherwise an operational risk report is prepared. Finally, a report summarizing the transaction and containing communication details with the client is generated automatically by a **system** role, the **senior finance manager** approves the transaction and the customer representative notifies the client.

Some observations about this process are as follows. First, this process is composed of individual tasks and subprocesses. Three key subprocesses are shown inside dotted-line boxes in Figure 2. These are the accounting entriy, authorization and risk evaluation subprocesses, and each is composed of atomic tasks. In general process design can be simplified by breaking down a process into subprocesses so that each subrpocess can be designed independently. Moreover, an OR-split node represents a choice or a decision point. Thus, both t3 (validate transfer instructions) and t4(check transaction limit) are decision points as are t6, t8 and t13. A parallel

structure is introduced in the subprocess “accounting entry”, where when funds availability test is passed, two entries, one for business accounting (t10a) and the other for any fee related to this transaction (t10b), are created in parallel and then merge in an AND-Join node.

In the example of Figure 2 we also show the task category to which a task belongs in a label at the top right of the task box, and the role that performs that task at the side. These concepts will be discussed next.

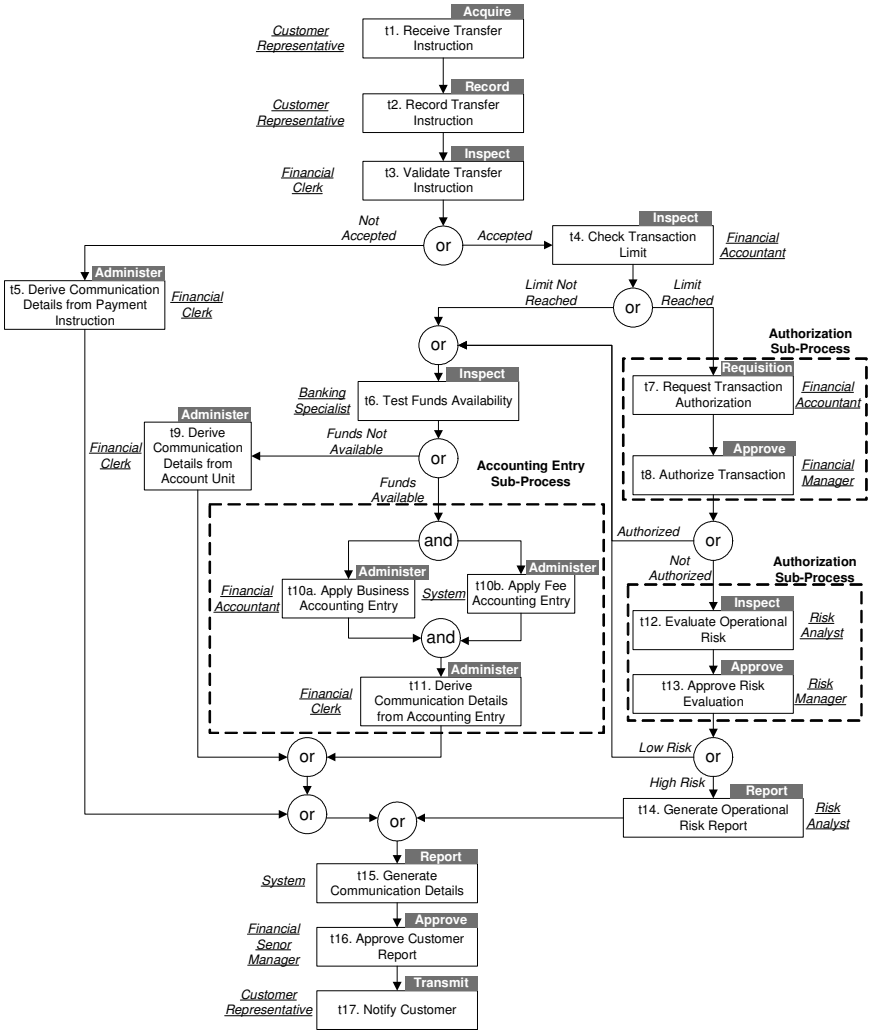


Figure 2: A formal representation of an account transfer request process

3 . Basic control requirements and principles

In this section we first introduce the notion of generic task categories and then discuss how control principles apply to various categories of tasks.

3.1 Task categories

A business process consists of tasks performed in a coordinated manner. In general, these tasks can be classified into certain categories of generic task categories. As a starting point for this work, we have developed 10 categories (see Table 1). These are categories were inspired by the Financial Services Workflow Model of IBM Information FrameWork, a comprehensive set of banking specific business models that represent best practice in banking industry [15]. We have modified them slightly to cover most business processes in other industries, and most business tasks can be, broadly speaking, classified into one of these categories.

Thus, in a generic sense *request, ask, initiate, order* are synonymous terms for *requisition*. Similarly, *authorize* is a generic term for approval or grant of permission for an order to be placed, a business trip to be taken, or a payment to be made. Similarly, *administer* is a generic term to cover a variety of tasks such as manipulate, move, inquire, search, etc. Thus, if, say in an order process, an order has been placed, but the goods have not been received on time and are overdue, an inquire step may be added to query the vendor, or escalate the order. It should also be noted that some tasks are more sensitive than others such as those involving transfer to goods or money. Thus, approve, inspect, transmit are more sensitive than requisition and administer. However, our framework treats all of them in the same way.

With respect to the example of Figure 2, each task in the process is labeled at the top right with the category to which it belongs. Thus, in this example there are tasks that belong to 8 of the 10 categories in Table 1.

Table 1: Generic task categories

Task category	Description
Prepare	Make something ready for use
Record	Note, enter into system, store in database
Approve	Accept, reject, decide, signoff
Requisition	Request, ask, initiate, order
Transmit	Notify, provide, deliver, send payment, goods etc. (outside the organization).
Acquire	Receive, obtain
Administer	Manipulate, move, inquire, search
Inspect	Test, evaluate, check
Suspend/Terminate	Hold, finish, complete, stop temporarily
Report	Prepare a report, or some kind of output

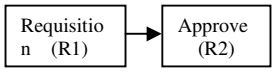
3.2 Security rules

At the outset it should be noted that control requirements are necessary in almost any business process where exchange of money or goods is involved. Moreover, this

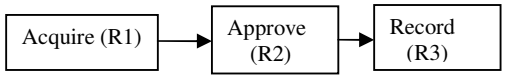
would apply regardless of whether a system is fully automated, partially automated, or entirely manual. In the automated case, the computer system should have been tested thoroughly before hand to make sure it operates correctly. In the manual case, the human worker must have the appropriate qualifications and authority to perform the task. In all cases, appropriate controls must be in place to prevent fraud or abuse of authority. Therefore, in this section we introduce basic control principles in a general manner. Subsequently, we will discuss how these principles can be operationalized.

The first standard principle of control (see Figure 3) is that a requester and approver for any task must be different [13]. This is the simplest situation of a separation of duties. Thus, a manager cannot approve her own expense claim for a business trip, but can do so for everybody else in the department. A further extension is the "3-eyes" rule. This requires separation of custody, approval and recording functions. Thus, for receipt of goods from a vendor, physical custody is kept by one person, the approval of the receipt is given by another, and the recording of the receipt is done by a third person. This ensures that the person receiving the shipment does not record it incorrectly. By separating receipt from recording, chances of fraud at delivery time are reduced. Moreover, there may be an additional requirement that the three individuals performing these tasks must be from different roles (say, inventory clerk, department supervisor and accounting clerk, respectively).

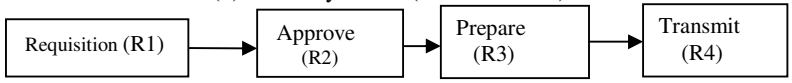
A further extension of this is the "four-eyes" rule which may require that for an order, the requester, authorizer, preparer of payment and the one of releases it, all be different individuals. For extra-sensitive transactions, multiple approvals may be required instead of one at each stage, for example by having two approvers (say, a manager and a VP) instead of one.



(a) Basic separation of duty (R1 ≠ R2)



(b) Three-eyes rule (R1 ≠ R2 ≠ R3)



(c) Four-eyes rule (R1 ≠ R2 ≠ R3 ≠ R4)

Figure 3: Basic paradigms for process security

Additional controls can also be added as a form of a "belt-and-suspenders" approach such as:

— additional signatures on large payments
— added approvals for new vendors
— end of day review of all large payments
— strong physical and system access controls.

In addition to separation of role requirements, ordering restrictions on roles may also be imposed. Thus, it may be necessary that a superior role (such as a manager or vice-president) may perform a task after a subordinate role (such as engineer).

4. Role Patterns

4.1 Overview

A process description includes the tasks that are performed in a process and the order in which they are performed. Along with a process description, it is important to provide information about who will perform a task. In general, this is accomplished by means of *roles*. A role is an organizational position that is qualified to perform certain tasks. Thus, in order to perform a task, a user must belong to a certain role that is qualified and authorized for it. Examples of roles in any typical organization are manager, director, VP, secretary, CEO, etc. The permissions or "power" of a person depends on her role. Thus, a department manager may approve travel requests for the employees in her department, while the human relations manager may approve leave requests, and the technology manager may approve requests for computer purchases. For some requests multiple approvals may be required. Moreover, some individuals may also hold multiple roles, e.g., a person may be a department manager and also an electrical engineer. However, in many situations an individual can play only one role for a particular process instance. Thus, a manager in the role of *department employee* while submitting his travel expenses cannot later assume the role of *department manager*, and approve her own expenses. This is akin to the notion of separation of duties discussed earlier, and a scenario like this is forbidden by the organization policy in most companies.

Table 2: Proposed role patterns

RP#	Role Pattern (RP) Description	Formal Expression
1	A task *t* must have an associated role *r* and belong to a task category *TC*.	RP1(*t*, *r*, *TC*)
2	No pair of tasks can be done by the same role in a (sub) process *p*.	RP2(*p*)
3	No pair of tasks with relationship *Rel* can be done by the same role in a (sub) process *p*.	RP3(*p*, *Rel*)
4	No task pair from a pair of different sub-processes, say s*p1* and s*p2*, can be done by the same role.	RP4(*sp1*,*sp2*)
5	No pair of tasks in a (sub) process *p* within a task category *TC* can be done by the same role.	RP5(*p*, *TC*)
6	There must be a minimum of *N* tasks from category *TC* executed in any instance of (sub) process *p*.	RP6(*p*, *TC*, *N*)
7	If multiple tasks of task category *TC* are done in a (sub) process *p*, the *later* task must not be done by a *lower* role than an *earlier* task in *TC*.	RP7(*p*, *TC*)
8	If one or more tasks of task category *TC* are done in a (sub) process *p*, then at least one of these tasks must be done by *role_min* or higher.	RP8(*p*,*TC*, *role_min*)
9	A process *p* must contain at least *N* unique roles.	RP9(*p*, *N*)
10	A role *r* can perform a maximum of *N* tasks in (sub) process *p*.	RP10(*p*, *r*, *N*)

Consequently, *role patterns* are a means of enforcing organizational policy on processes. These patterns can apply to tasks, subprocesses, and also complete processes. We have identified several common patterns that are relevant in the context of designing secure business processes and in accordance with business practice. These patterns are shown in Table 2.

Pattern RP1 requires every task to belong to a task category and to have at least one associated role. Patterns RP2 through RP5 express *separation of duties* requirements in various forms. For instance, RP2 is very stringent, while RP3 is less stringent. RP4 can apply to a group of subprocesses. Thus, in the context of Figure 2 it is reasonable that a role that is involved in the authorization subprocess should not be involved in the risk evaluation subprocess. RP5 requires that any pair of tasks within a subprocess that belong to the same task category must be done by different roles. Patterns RP6 through RP8 relate to the number of tasks within a category (say APPROVE) required for a process, and the role requirements when such task are executed in sequence (i.e. a later approval must be by a higher role than an earlier approval). Finally, RP9 and RP10 impose minimum and maximum limits on the number of unique roles in a process, and the number of tasks a role can perform in a process.

Next we discuss an example to illustrate the use of role-patterns.

4.2 Example of Role Patterns

We illustrate the application of role patterns in the context of the "Administer Account Transfer Process" of Figure 2 described above. In this example there are 18 tasks that fall in 8 categories, and are performed by 9 roles that must interact in order to complete it. Let *p* denote the process of “Administer Account Transfer”. There are various rules that apply to these roles and are expressed by role patterns as follows:

RP1. Every task has an associated role and belongs to a task category

RP3(*p*, *iseq*). A pair of tasks with immediate sequence (denoted as *iseq*) relationship between them cannot be done by the same role. Thus, in this process if two tasks are in immediate sequence then the roles must be different.

RP4("Authorize", "Risk Management"). No task pair from a pair of different sub-processes, say s*p1* and s*p2*, can be done by the same role. Thus, no task pair from Authorize and Risk Management subprocesses can be done by the same role.

RP5(*p*, ‘APPROVE’), RP5(*p*, ‘INSPECT’). No pair of tasks in a process within a task category APPROVE and the task category INSPECT can be done by the same role, i.e. two or more inspections, and two or more approvals must be done by different roles.

RP6(*p*, ‘APPROVE’, 1). There must be a minimum of one APPROVE category tasks executed in any process instance. In our example, there must be one approval no matter what path is taken through the process.

RP7(*p*, ‘APPROVE’). If multiple APPROVE tasks are done in a (sub) process, the *later* task must not be done by a *lower* role than any *earlier* APPROVE task. In our example, the final approval task is by a senior finance manager. The earlier approvals are done by a finance manager and a risk manager.

RP7(*p*, ‘INSPECT’). If multiple INSPECT tasks are done in a (sub) process, the *later* task must not be done by a *lower* role than any *earlier* INSPECT task. In our

example, these three INSPECT tasks are done respectively by financial clerk and two higher roles, financial accountant and banking specialist.

RP8(*p*, 'APPROVE', 'senior financial manager'). If one or more task of task category APPROVE are done in a (sub) process, then at least one of these tasks must be done by the role senior financial manager.

4.3 An implementation of role patterns for the example

There are many possible ways to represent and implement these patterns. Here we demonstrate one formal way using basic predicates in the Prolog [6] style shown in Figure 4. These 10 predicates are like templates that can be tailored to any specific example application. Next we show how the role patterns can be applied to the account transfer application of Figure 2.

```
role_occurs(Proc,R) :- assign(T,R), contain(Proc,T).
rp1(Proc,T) :- task(T,_,_,_),contain(Proc, T),
               role(R,_,_),assign(T,R).
rp2(Proc, T1, T2,R) :- assign(T1, R), assign(T2, R),T1\==T2,
                       task(T1,_,_,_),contain(Proc, T1),
                       task(T2,_,_,_),contain(Proc, T2).
rp3(Proc, T1, T2,R) :- iseq(T1,T2), assign(T1, R), assign(T2, R),
                       task(T1,_,_,_),contain(Proc, T1),
                       task(T2,_,_,_),contain(Proc, T2).
rp4(T1,Proc1,T2,Proc2,R):- assign(T1,R), assign(T2,R),T1\==T2,
                           task(T1,_,_,_),contain(Proc1, T1),
                           task(T2,_,_,_),contain(Proc2, T2),
                           subprocess(Proc1,_),subprocess(Proc2,_),
                           Proc1\==Proc2.
rp5(Proc,T1,T2,R,TC) :- task(T1,_,TC,_),contain(Proc, T1),
                        task(T2,_,TC,_),contain(Proc, T2),
                        assign(T1,R), assign(T2,R),T1\==T2.
rp6(Proc,T1,T2,P,TC,M) :- path(Proc,T1,T2,[T1],P),
                          tc_occurs(P,TC,N),N<M.
rp7(Proc,T1,T2,TC) :- path(Proc,T1,T2,[T1],_),
                      task(T1,_,TC,_), task(T2,_,TC,_),
                      assign(T1,R1),assign(T2,R2),
                      role(R1,_,N1),role(R2,_,N2), N2<N1.
rp8(Proc,T1,T2,P,TC,X,R):- path(Proc,T1,T2,[T1],P),
                           tc_occurs(P,TC,N),N>=1, role(Y,R,_),
                           task(X,_,TC,_),assign(X,Y),member(X,P).
rp9(Proc,Roles,N) :- setof(R, role_occurs(Proc,R),Roles),
                     length(Roles,M), M>=N.
rp10(Proc,R,Tasks,N) :- setof(T,(assign(T,R),contain(Proc,T),
                        process(Proc,_)), Tasks),
                        length(Tasks,M), M=<N.
```

Figure 4: Generic Prolog predicates to represent the 10 role patterns

First we represent all facts related to workflows in a form of relational tables. The facts are listed in the Appendix 1. The facts consist of task, role, subprocess, task assignement and process pattern (iseq, choice, parallel and loop) predicates. We also

define *contain, path,* and *tc_occurs* predicates. For example, `iseq(T1,T2)` is used to determine whether two tasks, `T1` and `T2`, are executed in immediate sequence. Note, in Prolog, a variable is an identifier starting with a capital letter, while a constant is one starting with lower-case letter. Therefore, for the process of Figure 2**,** `iseq(T1,t2)` returns `T1=t1`. Appendix 1 also serves as a template for representing workflows as Prolog facts. For any application, one could use similar tables to represent the facts related to a process and then build a fact database in Prolog.

With these facts, we can build several predicates. For example, `contain(proc,t)` predicate can be used to test if a task `t` is contained in a (sub) process `proc`. `path(Proc,A,B,Visited,Path)` predicate calculates all possible paths between two nodes A and B, and represent these paths as lists. `tc_occurs(Path,TC,N)` gives the number of tasks (`N`) occurs within a task category `TC` in a path. `role_occurs(Proc,R)` tests whether role `R` executes some task in process `Proc`. Then, role patterns can be formulated with the help of these predicates.

`rp1(proc,T)` is the Prolog implementation of role pattern RP1. This rule returns all tasks which satisfy this pattern. Pattern RP2 may not be applicable to the whole process, but within the authorization subprocess (denoted as `p2`), we may require that each task is executed by a unique role. Therefore, we can use `rp2(p2,T1,T2,R)` to find task pairs (`T1,T2`) which are executed by the same role `R`. If such a pair is found, then RP2 is violated in this subprocess. `rp3(Proc,T1,T2,R)` returns any pair of sequential task (`T1,T2`) that are executed by the same role `R`, indicating the violation of role pattern RP3. For the example shown in Figure 2, after testing this predicate, we found three violations as shown below.

```
?- rp3(p,T1,T2,R).
 T1 = t1,            T1 = t3,            T1 = t4,
 T2 = t2,            T2 = t5,            T2 = t7,
 R = r1 ;            R = r2 ;            R = r3 ;
```

These violations can be solved by (1) changing roles; (2) merging each pair of task into one; or, (3) not applying role pattern RP5 to this process. From a managerial point of view, if two consecutive tasks have to be executed by the same role, merging these two tasks into one may reduce the handover and improve efficiency.

`rp4(T1,Proc1,T2,Proc2,R)` tests if two task `T1` from subprocess `Proc1` and task `T2` from another subprocess `Proc2` are executed by the same role. For example, this predicate can be used to check if any role can participate in both authorization and risk management subprocesses. `rp5(Proc,T1,T2,TC,R)` can be used select a task pair (`T1,T2`) within category `TC` is executed by the same role `R`. Thus, `rp5(p,T1,T2,approve,R)` can be issued to check if two APPROVE tasks are done by the same role. `rp6(Proc,T1,T2,P,TC,M)` can be used to find out an execution path between task `T1` and `T2` which contains fewer than `M` tasks of category `TC`. For example, `rp6(p,t1,t17,P,approve,1)` returns any execution instance without any approvals in the process of Figure 2.

`rp7(Proc,T1,T2,TC)` finds that any pair of sequential tasks (`T1,T2`) in category are executed in sequence but the `T2` is executed in a lower role than `T1`. For example, using `rp7(p,T1,T2,approve)`, we can detect an APPROVE task (`T2`) which is done by a lower role than any previous APPROVE task (`T1`).

`rp8(Proc,T1,T2,P,TC,X,R)` selects any task (`X`) that is executed by role `R` in a path (`P`) between `T1` and `T2` which contains multiple tasks of category `TC`. For example, `rp8(p,t1,t17,P, approve, X, 'Senor Financial Manager')` can be used to find the task which is executed by a senior finance manager in any execution path that has at least one approval.

`rp9(Proc,Roles,N)` is used to test whether a (sub)processes has at least `N` unique roles. For example, `rp9(Proc,Roles,5)` can find (sub)processes that require at least 5 unique roles. For the example of Figure 2, none of the subprocesses meets this requirement. `rp10(Proc,R,Tasks,N)` selects the roles in a (sub) process which execute at maximum `N` tasks. Using `rp10(p,R,Tasks,1)`, we can find the roles that executes only one task in the process of Figure 2. Note that both predicates use a Prolog built-in predicate "setof". This predicate yields collections for individual bindings of the variables in the goal. For example, `setof(R,role_occurs(Proc, R),Roles)` gives list of roles returned from predicate `role_occurs`.

4.4 An Overall Approach

Above we have developed a methodology to systematically manage controls in business processes. The main features of our approach are:

1) Tasks are organized into *generic* task categories
2) Basic *process patterns* are used to describe processes
3) Basic *Role patterns* are used to describe control requirements.
4) The role patterns are associated with a process as per the business policies.
5) The patterns are implemented in a logic-based software application (such as Prolog).
6) Before making any task assignment to a role, the software performs checks and disallows certain tasks if they violate the control requirements.

5. Discussion and Related Work

5.1 Importance of Controls

The Sarbanes-Oxley Act of 2002 applies to all companies traded on U.S. stock exchanges. It was enacted into law in response to the major financial scandals such as Enron, MCI, and others in recent years. The law imposes tough requirements and penalties to ensure that financial statements accurately represent the actual business position of a company. The two sections that are most relevant to our work are Sections 302 and 404 [13,14]:

- Section 302 states that CEOs and CFOs must personally sign off on their companies' financial statements. Few specific controls are required by Section 302. The main point of this section is to establish CEO/CFO accountability for the rest of the Act's sections, with the possibility of prison for noncompliance.

- Section 404 mandates that well-defined and documented processes and controls be in place for all aspects of a company's operations that affect financial reports. Furthermore, executive management and a company's auditors must each state in writing that these processes and controls have been examined and are effective. Any findings of ineffectiveness must be publicly disclosed. For companies whose

net worth exceeds $75 million, this rule went into effect beginning with fiscal years ending June 2004.

Clearly, internal controls are indispensable for enforcing the requirements of the new legislation. Moreover, they can help a company achieve its profit goals and performance targets. In recent years, numerous companies have invested thousands of additional staff hours in complying with the requirements of Sarbanes-Oxley. During this activity they have realized that poor documentation of financial controls is a common problem. As a byproduct, one side benefit of this new law has been a fervent effort by various companies to automate and standardize their financial processes [3]. Such streamlining of processes has resulted in considerably reduced risks of misstatements on financial reports.

Naturally, there is a need for formal frameworks for ensuring that appropriate controls are in place, and the pattern-based reasoning approach proposed in this paper fits well in this context. It offers considerable appeal for both its simplicity and practical value. Business process modeling allows business analysts to formally define a process to reflect the inner workings of a business. This exercise is formalized by using a standard methodology and a tool for business process modeling. There are several tools such as IBM Websphere Business Modeler [11] that allow a visual model of a business process to be built. However, most tools do not provide adequate support for security and this is often added in a piecemeal and rather adhoc manner. If companies have to incorporate security in their business processes in order to achieve Sarbanes Oxley compliance, then it will have to be done in a more systematic manner, not as an afterthought.

5.2 Architecture for adopting role patterns

Figure 5 gives an architecture for integrating role patterns into an existing workflow system. We propose the addition of a new module called the security requirement modeling, which would allow a user to describe their security needs. These needs are easily converted into role patterns which are stored in the database. Moreover, the process description schema in the existing process modeling system is also translated into process facts including task definition, role assignment and process patterns and stored in the database. This requires a translation program that can examine the current process schema and convert it into the new format of the process patterns. When a task assignment is made by the business process modeling system, it would call a query engine, which would run a query against the database to ensure that none of the constraints is violated. If there is a violation of any role pattern, then the query engine will prevent the assignment from being made and explain the violation to the user. Moreover, during the process execution, the process execution engine can also query the database to ensure that any changes to the role assignment made dynamically during the runtime (e.g. delegation or reassignment) still comply to the role patterns. Finally, security control at the user level (e.g., any user cannot play roles as both requester and approver) can also be added to the database and directly constrain the process execution.

Moreover, the translation from the process description in an existing workflow system to the process patterns does not have to be very precise. Workflow systems offer a variety of constructs or patterns to capture complex coordination requirements [1]. For purposes of enforcing the role patterns such precise translation of each construct is not required. Since the role patterns refer only to two process patterns,

sequence and parallel, the translation at a minimum only needs to capture the tasks in processes (and their subprocesses) with respect to just these two patterns. Therefore, the translation can be done efficiently in an approximate manner and the role patterns can still be verified.

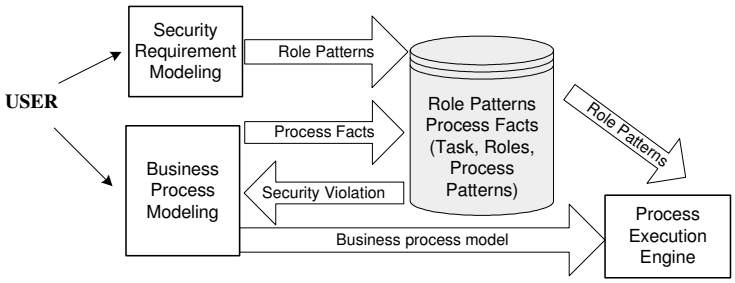


Figure 5: An architecture for adding security in existing workflow engine

5.3 Related Research

Prior research has looked at the issue of security from various perspectives. However, the stream of security related research that is most relevant here relates to role based access control (RBAC) and was pioneered by Sandhu [19]. The basic RBAC framework consists of three entities: *roles, permissions* and *users*. Roles (such as manager, director, etc.) are assigned permissions or rights (to hire an employee, approve a purchase, etc.) and users (Sue, Joe, Lin) are associated with roles. Thus, users acquire certain permissions to perform organizational tasks by virtue of their membership in roles. The notion of separation of duties [16,20], although it preexisted in accounting and control systems, also reemerged in the context of RBAC as the idea that if task 1 is performed by role A, then task 2 must be performed by role B, and membership of these roles must not intersect. There are two types of separation of duties: *static* and *dynamic*. In recent years, RBAC has become the preferred access control model for most business enterprises. This framework allows association of roles with tasks, and only users that belong to a certain role can perform certain tasks. This is a useful framework that has now been widely adopted in popular database management systems from IBM and Oracle.

Some related work on specification and enforcement of role-based authorizations in workflow systems is discussed in [4]. The main focus of this work is on enforcement of constraints at run-time. The authors develop algorithms to check whether, given a combination of tasks and users, it is possible to find a task assignment that will satisfy the collection of constraints and available users. A formal model called W-RBAC for extending RBAC in the context of workflows using the notions of case and organizational unit is described in [21]. A system architecture for enforcing RBAC in a Web-based workflow system is given in [2]. The approach in [5] is based on the notions of conflicting roles, conflicting permissions, conflicting users and conflicting tasks. More sophisticated algorithms for enforcing separation of duties in workflows are developed in [17]. Our work differs from and also complements previous works in that our focus is on role patterns, and our goal is to give end users the ability to associate one or more patterns with processes. Moreover, we have a more sophisticated process model than the ones used in previous works,

resulting in a tighter integration between the process model and the security model. Furthermore, our process model is not hardcoded into the constraints, unlike in previous models, and thus offers greater flexibility for associating tasks in a process with role patterns.

Another stream of prior work that informs our research is the literature on basic financial control principles, particularly as it relates to the recent Sarbanes-Oxley legislation [3,7,13,14]. Businesses have enforced financial controls for more than 100 years with the objective of preventing fraudulent activities and abuse of privilege to the detriment of the organization. In the past these financial controls were manual in nature, but in recent years the emphasis on automation of controls has increased considerably. While the Sarbanes-Oxley legislation does require businesses to certify that proper controls and processes are in place to prevent incidence of fraud by its managers, yet it does not specify the precise nature of such controls and processes. Many businesses have adopted ad hoc kinds of approaches to respond to this new law.

6. Conclusions

The focus of enterprise business process management lies in automating, monitoring and optimizing the information flow in an organization [9]. The recent Sarbanes-Oxley legislation in the aftermath of some high profile business scandals has created an even greater awareness of the importance of internal controls and the important role that streamlined processes play in implementing effective control systems. Internal controls are most effective when they are built into the enterprise infrastructure [7]. Therefore, internal controls must be tightly linked to business processes and companies are starting to realize the strategic value of making automated processes a part of daily business practice [3]. The focus of this paper is on creating a framework to embed controls into the processes of an organization. The main elements of this framework are process patterns, task categories, role patterns and constraints. We showed how a user can describe a process in a hierarchical manner using simple process patterns such as sequence, parallel, choice and loop, and then associate one or more of 10 standard role patterns with it to create a secure business process. We also showed that the role patterns can be implemented through built-in constraints. Although we did not discuss this at length, additional user-defined constraints can also be added for special requirements.

The key advantages of this approach are that it is generic, easy to use and flexible. The role patterns that capture common control requirements can be associated with process patterns in an easy and user-friendly manner. Finally, the role patterns are not hardcoded and can also be extended. In future work, we expect to implement this framework and test it in a real environment. We would also like to add temporal extensions. For instance, consider a policy like, "A manager cannot approve any requests until he has been in the manager role for 6 months." Here an individual may be in the manager role, but she may still not perform certain tasks. To handle such situations, one possibility is to create a special role called ‘New Manager’ and not associate it with certain tasks. However, more flexible ways of dealing with such situations are required. Finally, it would also be useful to consider issues of delegation [21], i.e. can a role delegate its tasks to other roles, and explore how organization policy on delegation could be incorporated into the process securely.

References

1. Aalst, W.M.P. van der, Hofstede, A.H.M. ter, Kiepuszewski, B., and Barro, A.P. "Workflow patterns," *Distributed and Parallel Databases,* 14(3):5-51, July 2003.
2. G.-J. Ahn, R. Sandhu, M.H. Kang, J.S. Park, Injecting RBAC to secure a web-based workflow system, in: Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany, July 2000. http://citeseer.nj.nec.com/ahn00injecting.html.
3. D. Berg, "Turning Sarbanes-Oxley Projects into Strategic Business Processes," *Sarbanes-Oxley Compliance Journal*, November 2004.
4. E. Bertino, E. Ferrari, V. Atluri, The specification and enforcement of authorization constraints in workflow management Systems, *ACM Trans. Inf. Syst. Secur.* 2 (1) (1999) 65–104.
5. Botha, R. A. and J. H. P. Eloff: 2001, 'Separation of duties for access control enforcement in workflow environments'. *IBM Systems Journal* 40(3).
6. W. F. Clocksin , C. S. Mellish, *Programming in Prolog*, Springer-Verlag New York, Inc., New York, NY, 1987.
7. Committee of Sponsoring Organizations, Internal Control – Integrated Framework, http://www.coso.org/publications/executive_summary_integrated_framework.htm.
8. R. Eshuis, R. Wieringa: Verification Support for Workflow Design with UML Activity Graphs, in the Proceedings of the 24th *International Conference on Software Engineering*, Orlando, Florida, May 19-25, 2002, pp. 166-176.
9. D. Ferguson and M. Stockton, "Enterprise Business Process Management - Architecture, Technology and Standards," *Business Process Management* 2006, Vienna, Austria, 1-15.
10. Gamma, Erich; Richard Helm, Ralph Johnson, *and* John Vlissides *(1995).* http://www.awprofessional.com/title/0201633612*Design Patterns: Elements of Reusable Object-Oriented Software, hardcover, Addison-Wesley.*
11. IBM Websphere Business Modeler (WBM), Version 6, http://www-306.ibm.com/software/integration/wbimodeler/.
12. Information FrameWork (IFW), IBM Industry Models for Financial Services, http://www-03.ibm.com/industries/financialservices/doc/content/bin/ fss_ifw_gim_2006.pdf
13. Scott Green, Manager's Guide to the Sarbanes-Oxley Act: Improving Internal Controls to Prevent Fraud, Wiley, 2004.
14. D. Haworth and L. Pietron, Sarbanes-Oxley: Achieving Compliance by Starting with ISO 17799, *Information Systems Management*, Winter 2006.
15. W-K. Huang, V. Atluri, Secureflow: a secure web-enabled workflow management system, in: Proceedings of the Fourth *ACM Workshop on Role-Based Access Control*, 1999, pp. 83–94.
16. D. R. Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems," *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Fairfax, VA (October 1997), pp. 23–30.
17. Liu, D., Wu, M., and Lee, S. 2004. Role-based authorizations for workflow systems in support of task-based separation of duty. *J. Syst. Softw.* 73, 3 (Nov. 2004), 375-387.
18. N. Nagaratnam, A. Nadalin, M. Hondo, M. McIntosh, and P. Austel, Business-driven application security: From modeling to managing secure applications, *IBM Systems Journal*, Volume 44, Number 4, 2005.
19. R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, *IEEE Computer* 29 (2) (1996) 38–47.
20. R. Simon and M. E. Zurko, "Separation of Duty in Role-Based Environments," *Proceedings of the 10th Computer Security Foundation Workshop*, Rockport, MA (June 10–12, 1997), pp. 183–194.
21. J. Wainer, A. Kumar and P. Barthelmess, "DW-RBAC: A Formal Security Model of Delegation and Revocation in Workflow Systems," *Information Systems*, Volume 32, Issue 3, May 2007, Pages 365-384.

Appendix 1 – Facts and basic predicates for describing the Account Transfer process

```
%% process table (id, name)
process(p1,'administer account transfer').
process(p2,'authorization sub-process').
process(p3,'accounting entry sub-process').

%% subprocess table (id, parent (sub) process id)
subprocess(p2,p1).
subprocess(p3,p1).

%% task table (id, name, task category, (sub) process)
task(t1, 'receive transfer instruction',acquire,p1).
task(t2, 'receive transfer instruction',record,p1).
task(t3, 'validate transfer instruction',inspect,p1).
……
%% iseq table (task id, task id) - a pair of task in immediate
sequence
iseq(t1, t2).
iseq(t2, t3).
……
%% choice table (task id, taskid)
choice(t4,t5).
……
%% define parallel tasks or blocks
parallel(t10a, t10b)

%% defin tasks or blocks in a loop. There is no loop in this example
%% loop(x, y)

%% role table (id, name, authority level)
role(r1, 'customer representative',0).
role(r2, 'financial clerk',0).
role(r3, 'financial accountant',1).

%% assignment table (task id, role id)
assign(t1, r1).
assign(t2, r1).

contain(Proc,Proc).
contain(Proc,T) :- task(T,_,_,Proc).
contain(Proc,Subproc) :- subprocess(Subproc,Proc).
contain(Proc,Subproc) :- subprocess(Subproc,X),X\==Proc,
                         contain(Proc,X).
contain(Proc,T) :- task(T,_,_,Subproc),Subproc\==Proc,
                   contain(Proc,Subproc).

path(Proc,A,B,Visited,Path) :- iseq(A,B),append(Visited,[B],Path),
                               contain(Proc,A), contain(Proc,B).
path(Proc,A,B,Visited,Path) :- iseq(A,C),C \== B,\+member(C,Visited),
                              append(Visited,[C],Q),
                               path(Proc,C,B,Q,Path),
                               contain(Proc,A),contain(Proc,B),
                               contain(Proc,C).

tc_occurs([], _, N) :- N is 0.
tc_occurs([H|T], TC, N):- tc_occurs(T, TC, M),task(H,_,TC,_),N is M+1.
tc_occurs([H|T], TC, N) :- tc_occurs(T, TC, N),\+task(H,_,TC,_).
role_occurs(Proc,R) :- assign(T,R), contain(Proc,T).
```