10

EXTENDING THE BFA WORKFLOW AUTHORIZATION MODEL TO EXPRESS WEIGHTED VOTING

Savith Kandala and Ravi Sandhu

Abstract Bertino, Ferrari and Atluri (BFA) have recently presented a model for specifying and enforcing authorization constraints for Workflow Management Systems (WFMS). The model is comprehensive and exhibits strong properties such as (1) a language to express constraints, (2) formal notions of constraint consistency and (3) algorithms for role-task and user-task assignments. In this paper, we extend the BFA model to include primitives for weighted voting. We show that the BFA model cannot express weighted voting in a straightforward manner, whereas Transaction Control Expressions (TCEs) proposed by Sandhu [5] incorporates this notion. Since, all other aspects of TCEs can be easily simulated in BFA, we believe that the notion of weighted voting is a fundamental operation which is missing in the BFA model. Although, we do not formally prove that BFA cannot simulate weighted voting, we make a strong case that this cannot be done easily or directly. We also show that the extended-BFA model retains all the strong properties of the BFA model.

Keywords: Access control, authorisation constraints, workflow authorizations

1. INTRODUCTION

In recent years, workflow management systems (WFMSs) have gained popularity both in research and commercial sectors. WFMSs are used to coordinate and streamline business processes of an enterprise. The security requirements imposed by these workflow applications calls for suitable access control mechanisms. An access control mechanism enforces the security policy of the organization.

Bertino, Ferrari and Atluri (BFA) have recently presented a model for specifying and enforcing authorization constraints for Workflow Management Systems (WFMS) in [1]. The model is comprehensive and exhibits strong properties such as (1) a language to express constraints, (2) formal notions of constraint consistency and (3) algorithms for role-task and user-task assignments. In this paper, we try to express a much older model called Transaction Control

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35508-5_22

V. Atluri et al. (eds.), *Research Advances in Database and Information Systems Security*

Expressions (TCEs) in the BFA model. TCEs were proposed by Sandhu [5], for the purpose of enforcing dynamic separation of duties constraints. The BFA model has a much wider scope than TCEs. So it is natural to ask whether or not BFA can simulate TCEs.

In spite, of the generality of BFA we show in this paper that the BFA model cannot fully express TCEs. In particular, the notion of weighted voting, which is a component of TCEs, cannot be expressed in the BFA model. Our paper does not prove this formally, but it does make a compelling case that there is no straightforward way of expressing weighted voting in BFA. So at least from a pragmatic viewpoint BFA should be extended to include weighted voting. We also show that the strong properties of the BFA model are still preserved in the extended-BFA model, which incorporates weighted voting.

The rest of the paper is organized as follows. Sections 2 and 3 give an overview of TCEs and the BFA model respectively. Section 4, shows how most aspects of TCEs can be easily expressed in the BFA model, except for weighted voting. Section 5 gives the extended-BFA model and section 6 shows that the properties of the BFA model are still retained in extended-BFA model. Section 7 gives the conclusion.

2. TRANSACTION CONTROL EXPRESSIONS (TCE'S)

In this section we describe the Transaction Control Expressions as proposed by Sandhu in [5]. The mechanism is very natural and intuitive, being close in spirit and letter to controls typically encountered in paper-based systems. In fact, it reflects the world of forms and books in the electronic media of databases.

Example 2.1

Consider a situation in which payment in the form of a check is prepared and issued by the following sequence of events.

1. A clerk prepares the check.
2. A supervisor approves the check.
3. A clerk issues the check.

We can specify some separation of duties constraints so that the users cannot perpetrate fraud in the system. One such constraint can be explicitly stated as, the users performing prepare, approve and issue transactions should all be different. So it will take collusion of two clerks and a supervisor to perpetrate fraud. This is called dynamic separation of duties since a clerk can perform steps 1 and 3 on different vouchers, but not on the same one. Static separation of duty would identify two kinds of clerk roles, say preparation_clerk and issue_clerk, where the former can only do step 1 and the latter only do step 3. Clearly dynamic separation of duties is more efficient.

The above example is expressed in TCEs as follows:

prepare • clerk;
approve • supervisor;
issue • clerk;

Each term of a transaction control expression has two parts. The first part names a transaction. A user assigned (explicitly or implicitly[1]) to the *role* specified in the second part can execute the transaction.

The term "prepare • clerk;" specifies that the prepare transaction can be executed on a check object only by a clerk. The semi-colon signifies sequential application of the terms. That is a supervisor can execute the approve transaction on a check only after a clerk has successfully executed the proceeding prepare transaction. Finally, separation of duties is further enforced by requiring that the users who execute different transactions in the transaction control expression all be distinct. As individual transactions are executed the expression gets incrementally converted to a history, for instance as follows.

prepare • *Alice*;	prepare • *Alice*;	prepare • *Alice*;
approve•supervisor;	approve•*Bob*;	approve•*Bob*;
issue•clerk;	issue•clerk;	issue•*Chris*;
(a)	(b)	(c)

The identity of the user who executes each transaction is recorded to enforce the requirement that these users be distinct. So if *Alice* attempts to issue that check after point (b) in this sequence the system rejects the attempt.

A transaction control expression thus contains a history of transactions executed on the object in the past and a potential history, which authorizes transactions that can be executed in the future. The expression begins as a constraint and ends as a complete history of the object. In a manual system identification of the users executing each transaction is achieved by signatures. In automated systems user identities must be recorded with guaranteed correctness.

Sometimes, different transactions in an object history must be executed by the same user.

Example 2.2

Consider the following scenario, a project leader initiates a requisition, a purchase order is prepared for the requisition by a clerk and approved by a purchasing manager. The purchase order then needs agreement of the same project leader, who was involved in requisition. Finally, the clerk places the order.

The following syntax was proposed to identify steps must be executed by the same user.

requisition •project_leader ↓ x;
prepare_order •clerk;
approve_order •purchasing_manager;
agree • project_leader ↓ x;
order•clerk;

The anchor symbol "↓" identifies which steps must be executed by the same individual. The "x" following it is a token for relating multiple anchors. For instance, in the above TCE if the same clerk had to prepare the purchase order

[1] Descriptions of role hierarchies and implicit versus explicit role assignments are presented in [6].

and place the order then we can use the anchor symbol "↓" with a token "y" for the second and fifth terms in the above TCE.

In some cases, any authorized user can execute a transaction in an object history. We modify Example 2.1. Any authorized user assigned to the supervisor role can perform step 2. It would still take a collusion of a supervisor and a clerk to perpetrate fraud in the system. If the role hierarchy is such that the supervisor is senior to the clerk, then the supervisor can perform prepare and approve or he/she can perform approve and issue. But, the supervisor cannot perform all the three steps.

The following syntax identifies steps, which can be performed by any authorized user.

prepare • clerk;
approve • supervisor ↑;
issue • clerk;

Since it is a free step a token to identify multiple anchors is not needed. (Free steps are an extension to the TCE mechanism proposed in [5].)

We now turn the focus on a voting scheme scenario. In some cases, any authorized user can execute a transaction in an object history. We modify Example 2.1. The second step now requires that three different supervisor approve the check. The following syntax is used to express the voting scheme.

prepare • clerk;
3: approve • supervisor=1;
issue • clerk;

The colon is a voting constraint specifying 3 votes from 3 different supervisors. This notion is further extended to include weights of different role as follows:

prepare • clerk;
3: approve • manager = 2, supervisor = 1;
issue • clerk;

In this case, approve transactions with sufficient votes are required before proceeding to the next term. The number of votes required is interpreted as a lower bound. The moment 3 or more votes are obtained the next step is enabled.

3. THE BFA WORKFLOW AUTHORIZATION MODEL

In this section we describe the workflow authorization model proposed by Bertino, Ferrari and Atluri in [1], which for convenience we call the BFA model. The BFA model gives a language for defining constraints on role assignment and user assignment to tasks in a workflow. By using this language, we can express conditions constraining the users or roles that can execute a task. The constraint language supports, among other functions, both static and dynamic separation of duties. Because, the number of tasks and constraints can be very large, the BFA model provides formal notions of constraint consistency and has algorithms for consistency checking. Constraints are formally expressed as clauses in a logic

program. The BFA model also gives algorithms for planning role and user assignments to various tasks. The goal of these role-task and user-task planners is to generate a set of possible assignments, so that all constraints stated as part of authorization specification are satisfied. The planner is activated before the workflow execution starts to perform an initial plan. This plan can be dynamically modified during the workflow execution, to take into account specific situations, such as the unsuccessful execution (abort) of a task.

In the BFA model, as in most WFMSs, there is an assumption that a workflow consists several tasks to be executed sequentially. A task can be executed several times within the same workflow. Such an occurrence of a given task *T* is called an *activation* of *T*. All activations of a task must be complete before the next task in the workflow can begin. Each task is associated with one or more roles. These roles are the only ones authorized to execute the task. In the remainder of this section **U, R, T** respectively denote the set of users, the set of roles and the set of tasks in a given workflow.

In the BFA model the workflow role specification is formally defined as follows.

Definition 3.1 (BFA Workflow Role Specification) A workflow role specification *W* is a list of task role specifications [TRS_1, TRS_2,...,TRS_n], where each TRS_i is a 3-tuple (T_i, (RS_i,$>_i$), act_i) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i , $>_i$, is a local role order relationship, and $act_i \in$ **N** is the number of possible activations of task T_i. The workflow tasks are sequentially executed according to the order in which they appear in the workflow role specification.

In order to provide a semantic foundation for the BFA model and to formally prove consistency, the constraints are represented as clauses in a normal logic program. The clauses in a logic program can contain negative literals in their body.

Definition 3.2 (Constraint Specification Language) The constraint specification language was specified by defining the set of constants, variables and predicate symbols.

A complete list of all the predicates and their descriptions is given in [1].

A rule is an expression of the form:

H ← A1,..... An, not B1,.....not Bm, n,m>= 0

Where H, A1,.....An and B1.....Bm are atoms and not denotes negation by failure. H is the head of the rule and whereas A1,.....An, not B1.....not Bm is the rule body. Rules can be expressed in the constraint specification language can be classified into a set of categories according to the predicate symbols they contain. Namely, explicit rules, assignment rules, static checking rules and integrity rules. The definitions of these rules and their description are given in [1].

Definition 3.3 (Constraint-Base) Let W be a workflow. The Constraint-Base associated with W (written CB (W)) consists of a set of explicit, assignment and integrity rules.

Intuitively, a CB is consistent if and only if the constraints it encodes are satisfiable. The consistency of a CB is determined by computing and analyzing its model. Details on CB consistency, consistency analysis and role-task / user-task assignment algorithms are presented in [1].

4. EXPRESSING TCE'S IN BFA

In this section we show how the separation of duties constraints of TCEs can be expressed in the BFA model. We illustrate this by expressing all the TCEs mentioned in section 2 in BFA. In this section we also argue that the weighted voting scenario which was expressed in section 2 cannot be expressed in BFA. Although, this is not formally proved, we make a compelling case that there is no straightforward way of expressing weighted voting in BFA. So at least from a pragmatic viewpoint BFA should be extended to include weighted voting.

Basic TCE

Consider the Basic TCE presented in section 2 and the global role hierarchy where the supervisor dominates clerk:

prepare • clerk;
approve • supervisor;
issue • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully executed prepare.

C_2: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 1),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB (W):

$R_{1,1}$: $\text{cannot\_do}_u(U, \text{approve}) \leftarrow \text{execute}_u(U, \text{prepare}, 1)$;

$R_{2,1}$: $\text{cannot\_do}_u(U, \text{issue}) \leftarrow \text{execute}_u(U, \text{approve}, 1)$;

$R_{2,2}$: $\text{cannot\_do}_u(U, \text{issue}) \leftarrow \text{execute}_u(U, \text{prepare}, 1)$;

TCE with Anchors

Consider the TCE with Anchors presented in section 2 and the global role hierarchy in Figure 10.2:

requisition •project_leader ↓ x;
prepare_order •clerk;
approve_order •purchasing_manager;
agree • project_leader ↓ x;
order•clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute prepare_order, if he/she had successfully executed requisition.

C_2: A user cannot execute approve_order, if he/she had successfully executed requisition or prepare_order.
C_3: A user executing agree must be the same user who had successfully executed requisition.
C_4: A user cannot execute order, if he/she had successfully executed requisition, prepare_order, approve_order or agree.

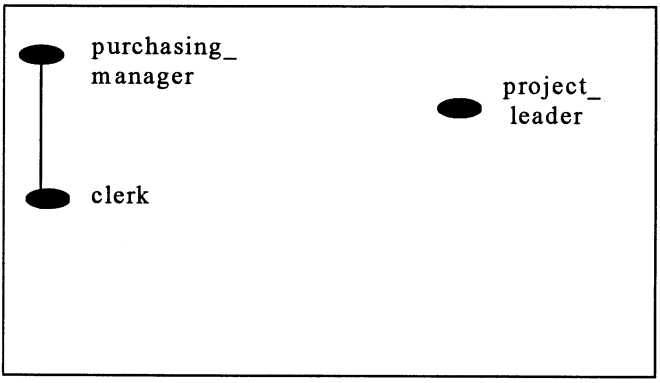

Figure 10.1. Role hierarchy.

These can be expressed in BFA as follows:
Workflow W = [(requisition,({project_leader}, {}), 1),
(prepare_order,({clerk, purchasing_manager}, {}), 1),
(approve_order,({purchasing_manager}, {}), 1),
(agree,({project_leader}, {}), 1),
(order,({clerk, purchasing_manager}, {}), 1)]
Constraint Base CB (W):
$R_{1,1}$: cannot_do$_u$(U, prepare_order) ← execute$_u$(U, requisition,1);
$R_{2,1}$: cannot_do$_u$(U, approve_order) ← execute$_u$(U, prepare_order,1);
$R_{2,2}$: cannot_do$_u$(U, approve_order) ← execute$_u$(U, requisition,1);
$R_{3,1}$: must_execute$_u$(U, agree) ← execute$_u$(U, requisition,1);
$R_{4,1}$: cannot_do$_u$(U, order) ← execute$_u$(U, agree,1);
$R_{4,2}$: cannot_do$_u$(U, order) ← execute$_u$(U, approve_order,1);
$R_{4,3}$: cannot_do$_u$(U, order) ← execute$_u$(U, prepare_order,1);
$R_{4,4}$: cannot_do$_u$(U, order) ← execute$_u$(U, requsition,1);

TCE with Free Steps

Consider the following TCE presented in section 2 and the global role hierarchy in Figure 10.1:

prepare • clerk;
approve • supervisor ↑;
submit • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 1),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB (W):

$R_{1,1}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve,1);

$R_{1,2}$: cannot_do$_u$(U, issue) ← execute$_u$(U, prepare,1);

TCE with Simple Voting

Consider the TCE with Simple Voting presented in section 2 and the global role hierarchy in Figure 10.1. In this TCE, equal weights are assigned to the role supervisor.

prepare • clerk;
3: approve • supervisor=1;
issue • clerk;

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully executed prepare.

C_2: A user cannot execute approve, more than once. (Three different users have to execute approve).

C_3: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in BFA as follows:

Workflow W = [(prepare,({clerk, supervisor}, {}), 1),
(approve,({supervisor}, {}), 3),
(issue,({clerk, supervisor}, {}), 1)]

Constraint Base CB(W) :

$R_{1,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, prepare,1);

$R_{2,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve,1);

$R_{2,2}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve,2);

$R_{3,1}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve,1);

$R_{3,2}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve,2);

$R_{3,3}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve,3);

$R_{3,4}$: cannot_do$_u$(U, issue) ← execute$_u$(U, prepare,1);

$R_{3,5}$: cannot_do$_u$ (U, issue) ← count(success(approve, k), n), n < 3;

TCE with Weighted Voting

Consider the TCE with Weighted Voting presented in section 2 and the role hierarchy in Figure 10.2. In this, TCE the notion of voting was further extended to include different weights to different roles as follows:

prepare • clerk;
3: approve • manager = 2, supervisor = 1;
issue • clerk;

In this case, the manager has a weight of 2 and the supervisor has a weight of 1 for the approve transactions. The number of votes required for approve transaction to be successful is 3. As soon as 3 or more votes are obtained the next step is enabled.

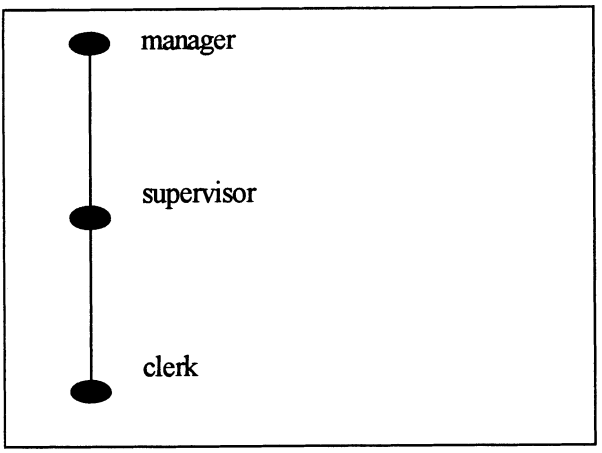

Figure 10.2. Role hierarchy.

We know that the BFA model does not support assigning weights to roles (by definition 3.1). So we enumerate the various possible role assignments to approve and try to capture all the possible assignments as constraints in CB (W). Table 10.1 below, gives the various possible assignments for approve.

Table 10.1. Possible role assignments for approve.

Possibility No.	Activation 1	Activation 2	Activation 3
1	Supervisor	Supervisor	Supervisor
2	Supervisor	Supervisor	Manager
3	Supervisor	Manager	-
4	Manger	Supervisor	-
5	Manger	Manager	-

Recall that, a CB for a workflow consists of a set of explicit, assignment and integrity rules (definition3.3). We now try to express all the possible role assignments for approve with these rules.

Expressing the possible role assignments as constraints using Explicit rules: Explicit rules contain an execution or a specification atom in the head and an empty body. The various possibilities of table 10.1 cannot be expressed using $execute_r$ as each activation of approve can be executed by the supervisor or the manager. We cannot use the constraints expressed as

$execute_r$ (supervisor, approve, 1) ←

or

$execute_r$ (manager, approve, 1) ←

to capture the role assignments of table 10.1 since, activation 1 of approve can be executed by the supervisor or manager. So we cannot express the possible role assignments for approve in table 10.1 using explicit rules.

Expressing the possible role assignments as constraints using Assignment rules:

Assignment rules contain a must_execute$_u$, must_execute$_r$, cannot_do$_u$ or cannot_do$_r$ atom in the head and specification atoms, execution atoms, comparison literals, or aggregate atoms in the body. The must_execute$_r$ or cannot_do$_r$ atoms can be used in the head of the rules to express the role assignment constraints. As each activation of approve can be assigned to a manager or a supervisor we cannot use the constraints expressed as

cannot_do$_r$(supervisor, approve) ← execute$_r$(supervisor,approve, 1), execute$_r$(supervisor, approve, 2) ;

or

must_execute$_r$(supervisor, approve) ← execute$_r$(supervisor, approve, 1), execute$_r$(supervisor, approve, 2) ;

to capture the role assignments of table 10.1. This is because, activation 3 of approve can be executed by the supervisor or manager, even if the activation 1 and activation 2 are successfully executed by the supervisor role. So we cannot express the possible role assignments for approve in table 10.1 using assignment rules.

Expressing the possible role assignments as constraints using Integrity rules: Integrity rules contain a panic atom in the head and specification atoms, execution atoms, comparison literals, or aggregate atoms in the body. The same argument presented for assignment rules holds here. So we can say, that we cannot express the possible role assignments for approve in table 10.1 using integrity rules.

We have already argued above, that the possible role assignments for weighted voting scenario cannot be expressed by the explicit, assignment and integrity rules of the CB. Also, the definition for a Workflow (definition 3.1) does not support assigning weights to each role. So, we conjecture there are no straightforward ways to express weighted voting schemes in BFA.

Conjecture 1: There are no straightforward ways to express weighted voting in BFA.

A stronger conjecture would assert that there are no ways, straightforward or convoluted, to express weighted voting in BFA. A formal proof of this stronger conjecture would be of theoretical interest but is outside the scope of this paper.

From a practical perspective it is quite simple to add the weighted voting feature to BFA. Weighted voting has been previously proposed in the literature and is intuitively a simple and natural concept. This is accomplished in the next section.

5. THE EXTENDED-BFA MODEL

In this section we describe the extended-BFA model, to accommodate the scenarios of weighted voting. We modify the definition of the BFA workflow-role specification (Definition 3.1) as follows.

Definition 5.1 (Extended-BFA Workflow Role Specification) A workflow role specification W is a list where each element in the list is either a task-role

specification or a vote-role specification [TRS_1/ VRS_1, TRS_2/ VRS_2 ……,TRS_n /VRS_n], where each TRS_i is a 3-tuple (T_i, (RS_i,$>_i$), act_i) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i, $>_i$, is a local role order relationship, and $act_i \in$ **N** is the number of possible activations of task T_i and each, VRS_i is a 4-tuple (T_i, (RS_i,$>_i$),*VotesRequired, RoleWeight*) where $T_i \in$ **T** is a task, $RS_i \in$ **R** is the set of roles authorized to execute T_i , $>_i$, is a local role order relationship, and *VotesRequired* $\in$ **N** is the number of votes required to make of task T_i and *RoleWeight* is a function that maps each role in the set RS_i to a *weight* $\in$ **N** for a given T_i. *RoleWeight:* $RS_i \rightarrow$ **N**

The workflow tasks are sequentially executed according to the order in which they appear in the workflow role specification. We propose two possible solutions for expressing the weighted voting constraints in extended-BFA model.

Solution 1 for expressing the weighted voting constraint in extended-BFA: The weighted voting constraint could be expressed as in terms of the number of successful activations of the task *approve* as follows:

cannot_do$_u$(U, issue) ← count(success(approve, k), execute$_r$(R_i, approve, k),R_i = manager, n_1),
count(success(approve,k), execute$_r$(R_j, approve, k),R_j = supervisor, n_2),
(2* n_1 + n_2) < 3;

Here, we count the number of successful executions of the task *approve* by the role manager (returned as n_1), the number of successful executions of the task *approve* by the role supervisor (returned as n_2) and compute the votes registered as (2* n_1 + n_2). Since the manager has a weight of 2 and the supervisor has a weight of 1. If the votes registered are less than 3, then users cannot perform the task *issue*.

This solution is inefficient if there are a number of different roles assigned different weights for the task. Therefore, we propose a second solution to express weighted voting efficiently in the extended-BFA model.

Solution 2 for expressing the weighted voting constraint in extended-BFA: We add the following weighted voting predicates and the weighted voting rule to efficiently express weighted voting in the extended BFA model.

Table 10.2. Weighted voting predicates.

Predicate	Arity	Arguments' Type	Meaning
role_weight	3	RT,TT,NT	role_weight(R_i, T_j, n) gets the weight of the role R_i assigned to the Task T_j and returns this value as n.
votes_required	2	TT,NT	votes_required(T_i, n) gets the minimum number of votes required for the task T_i to be successful and returns this value as n.
count_votes	2	TT,NT	count_votes(T_i, n) computes the following:

Predicate	Arity	Arguments' Type	Meaning

			for each $R_i \in RS_i$
			begin
			sum:=0
			count(success(T_i, k),
			execute$_r$(R_i, T_i,, k), n_1)
			role_weight(R_i, T_i, n_2)
			sum := sum + (n_1 * n_2)
			end;
			and returns the value of sum as n.

Definition 5.2 (Extended-BFA Constraint Specification Language)
The extended-BFA Constraint Specification Language is unchanged from definition 3.2 except for the addition of the weighted voting predicates described in table 10.2.

Table 10.3. Weighted voting rule.

Rule	Head	Body
WeightedVoting	cannot_do$_u$	conjunction of comparison literals and weighted voting predicates

Definition 5.3 (Constraint-Base) The extended-BFA Constraint-Base is unchanged from definition 3.3 except for the addition of the weighted voting rule described in table 10.3.

With the help of above weighted voting predicates and weighted voting rule we can now express a TCE with a voting scheme in Extended-BFA. Consider the TCE with Weighted Voting presented in section 2 and the global role hierarchy in Figure 10.2.

The separation of duties constraints can be enumerated as follows:

C_1: A user cannot execute approve, if he/she had successfully executed prepare.

C_2: A user cannot execute approve, more than once. (Three different users have to execute approve).

C_3: A user cannot execute issue until approve transaction registers the required number of votes.

C_4: A user cannot execute issue, if he/she had successfully executed prepare or approve.

These can be expressed in Extended-BFA as follows:

Workflow W = [(prepare,({clerk, supervisor, manager}, {}), 1),
(approve,({supervisor,manger},{}),3,{(supervisor,1),(manager,2)}),
(issue,({clerk, supervisor, manager}, {}), 1)]

Constraint Base CB(W) :

$R_{1,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, prepare,1);

$R_{2,1}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve, 1);

$R_{2,2}$: cannot_do$_u$(U, approve) ← execute$_u$(U, approve, 2);

$R_{3,1}$: cannot_do$_u$(U, issue) ← count_votes(T_I, n_1),votes_required(T_i, n_2), $n_1 < n_2$;

$R_{4,1}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 1);
$R_{4,2}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 2);
$R_{4,3}$: cannot_do$_u$(U, issue) ← execute$_u$(U, approve, 3);
$R_{4,4}$: cannot_do$_u$(U, issue) ← execute$_u$(U, prepare,1);

The rule $R_{4,3}$ will only be effective if the approve task had three activations (the maximum number of possible activations). If the number of activations for the approve task was less then three, then this rule will be ineffective as execute$_u$(U, approve, 3) will always be false.

6. PROPERTIES OF THE EXTENDED BFA MODEL

We now focus the attention on showing that the properties of the BFA model are still preserved in the extended-BFA model.

The properties of the BFA model as mentioned earlier are:

(1) a language to express constraints
(2) formal notions of constraint consistency and
(3) algorithms for role-task and user-task assignments.

In the extended-BFA model, we have not changed the language to express constraints, so property (1) is preserved. The formal proof for the following proposition as presented in [1] still holds.

Proposition 6.1 *Any CB is a stratified normal program. Hence, it has a unique stable model.*

Since we have not changed the definitions of explicit, assignment and integrity rules. The formal proof presented for this proposition in [1] still holds for these rules. We now extend the argument presented in [1] to include the weighted voting rule. A program P is stratified if its extended dependency graph does not contain any cycle involving an edge labeled with "not" [10], where the extended dependency graph of a program P is a graph whose nodes are the predicates that appear in the heads of the rules of P. Given two nodes p1 and p2 there is a direct edge from p1 to p2 if and only if predicate p2 occurs positively or negatively in the body of a rule whose head predicate is p1. The edge (p1, p2) is marked with a "not" sign if and only if there exists at least one rule r with head predicate p1 such that p2 occurs negatively in the body of r. By Definition 5.3, the CB associated with a given workflow consist of a set of explicit, assignment, integrity and weighted voting rules. The explicit, assignment and integrity rules cannot form a cycle in the extended dependency graph [1]. By definition the weighted voting rule has a planning predicate (cannot_do$_u$) as head and a conjunction of weighted voting predicates and comparison literals as body. Since, the predicates that appear in the head are disjoint from the predicates that can appear in the body, they cannot form any cycle in the extended dependency graph. Hence, the extended dependency graph associated with a CB does not contain any cycle. Thus, the CB is stratified.

The static analysis algorithm and the pruning algorithm need some modifications to accommodate the VRS_is in addition to the TRS_is. These modifications are simple to make in order to preserve property (2).

The role-task and user-task assignment algorithms also need modification, to accommodate the VRS_is. Instead, of looping through the number of activations in each TRS_i they have to also consider, that each element of the workflow can be a VRS_i. In case, an element is a VRS_i. they need to keep track of the number of votes required after each assignment and the weights assigned to each role. With these modifications, property (3) of the BFA model can also be preserved.

Thus, we argue that with some modifications to the static analysis algorithm, pruning algorithm, role-task assignment algorithm and user-task assignment algorithm the extended-BFA model preserves the strong properties of the BFA model.

7. CONCLUSION

In this paper, we have shown that the BFA model cannot be used to express the weighted voting scenario. We have extended the BFA model so that this feature can be accommodated in the BFA model. We have also argued that the extended-BFA model does preserve all the properties of the BFA model. It should also be noted that the constraint specification language of BFA is not intended for end-users to express constraints, it is rather used internally by the system to analyze and enforce constraints. The TCEs on the other hand are very natural and intuitive, so we can use TCEs as a language in which users can specify their separation of duties constraints. The constraints in TCEs can be translated to BFA. This reduction to extended BFA also provides a formal semantics, which has so far not been given.

Future directions of our research could involve developing an automated system to translate TCEs into the BFA model, this could be helpful as the BFA model has formal semantics for expressing constraints at the system level. The BFA model and the TCEs follow a strict sequence of execution. We would also like to introduce some parallelisms in the task execution.

References

[1] Bertino, E., Ferrari, E. and Atluri, V. (1997). A flexible model for the specification and enforcement of authorization constraints in workflow management system. *Proceedings of the Second ACM Workshop on Role-Based Access Control.*

[2] Clark, D. and Wilson, D. (1987). A comparison of commercial and military security policies. *Proceedings of IEEE Symposium on Security and Privacy*, pp. 184-194.

[3] Das, S. (1992). *Deductive Databases and Logic Programming*, Addison-Wesley.

[4] Nash, M. and Poland, K. (1987). Some conundrums concerning separation of duty. *Proceedings of IEEE Symposium on Security and Privacy*, pp. 201-207.

[5] Sandhu, R. (1988). Transaction control expressions for separation of duties. *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pp. 282-286.

[6] Sandhu, R., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (1996). Role-based access control models. *IEEE Computer*, **29(2)**, pp. 38-47.

[7] Sandhu, R. (1990). Separation of duties in computerized information systems. *Proceedings of the IFIP WG 11.3 Workshop on Database Security.*

[8] Simon, R.T. and Zurko, M.E. (1997). Separation of duty in role-based environments. *Proceedings of Computer Foundations Workshop X.*

[9] Thomas, R.K. and Sandhu R. (1997). Task-based authorization controls (TBAC) *Proceedings of the IFIP WG 11.3 Workshop on Database Security.*

[10] Ullman, J. (1989). *Principles of Database and Knowledge-Base Systems (2^{nd} Volume)*, Computer Science Press.